UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A120**

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

AUTOR: **CONSTANTINO BELLO CORBEIRA**

TUTOR: **JOSE LUÍS CALVO ROLLE**
OSCAR FONTENLA ROMERO

FECHA: **SEPTIEMBRE DE 2017**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

ÍNDICE GENERAL

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice de figuras

Índice de tablas

Listado de códigos de programación

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice del documento MEMORIA

1 Objeto

El presente proyecto tendrá como objeto diseñar, construir y configurar un sistema físico móvil formado por un sistema propulsor y un contrapeso, unidos entre sí por un eslabón rígido sujeto a un punto fijo. Dicho sistema constará de dos grados de libertad en el punto intermedio del eslabón (punto fijo), y un grado de libertad en el extremo propulsor. Además, se desarrollará un sistema complementario que permita la estabilización de dicho sistema físico en una posición mediante un algoritmo de control PID, que no será, y que se implementará en un sistema embebido.

2 Alcance

- Estudio, comparativa y elección del sistema embebido a emplear.
- Estudio de las técnicas de control PID aplicables al sistema.
- Estudio de los sensores (acelerómetro y/o giroscopio) que permitirán obtener la posición espacial del sistema objeto.
- Diseño y construcción del sistema objeto, una parte de este será impreso en 3D.
- Implementación de las técnicas de control en la placa seleccionada mediante el lenguaje de programación Python.

3 Antecedentes

El sistema físico objeto de la estabilización será diseñado desde cero, basándose en el diseño de un cuadricóptero, tanto a nivel estructural como de componentes, ya que se incluirán en este elementos característicos de un cuadricóptero como son los motores brushless y los reguladores de velocidad, conocidos mejor por sus siglas en inglés (ESC).

3.1. Sistemas Embebidos

En la fecha de planteamiento de este proyecto, se disponía de 3 sistemas embebidos, cada uno de ellos es apto para llevar a cabo la estabilización, a continuación se hablará de forma breve de dichos sistemas.

3.1.1. Arduino Mega 2560

En el año 2005, el estudiante del instituto IVRAE de Italia Massimo Banzi desarrolló una placa de microcontrolador de bajo coste para solventar el problema generado por el excesivo coste de las placas de microcontrolador, y ademas tratar de evitar la quiebra del instituto tecnológico en el que el estudiaba. El primer prototipo de arduino que se puede observar en la figura 3.1.1.1, era una placa formada por un microcontrolador muy simple que estaba conectado con un circuito de acondicionamiento, y al que únicamente se podían conectar sensores simples basados en resistencias y leds, además de que aún no contaba con un entorno de programación como el que nos resulta tan familiar en la actualidad. Posteriormente, estudiantes de diferentes nacionalidades fueron integrándose en el proyecto y mejorándolo, desarrollando el entorno de programación, mejorando la placa a nivel de hardware, agregando puertos USB que permitían conectarlo a un ordenador personal y, finalmente, distribuyendo Arduino a nivel mundial. A pesar de todo esto, Massimo Banzi fue incapaz de evitar la quiebra del instituto.

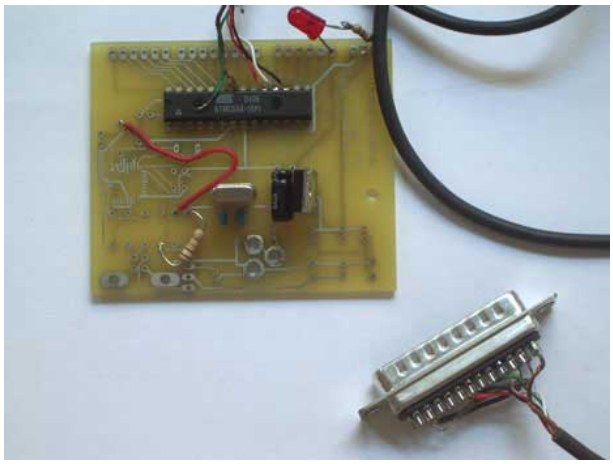


Figura 3.1.1.1 – Primer prototipo de Arduino (2005)

A lo largo de los años, diferentes modelos y revisiones de Arduino fueron saliendo al mercado, una de ellas es el Arduino Mega 2560 (Figura 3.1.1.2), la versión de la que se dispone, es un sistema embebido basado en el microcontrolador ATmega2560, cuenta con 54 pines de entradas/salidas digitales, de los cuales, 15 pueden ser utilizados como salidas PWM; 16 entradas analógicas y 4 UARTs. El microcontrolador funciona con una señal de reloj de 16MHz generada por un cristal incluido en la placa. Este sistema es fácilmente programable desde un Entorno de Desarrollo Integrado (IDE) creado y suministrado de manera gratuita por Arduino, el lenguaje de programación es un lenguaje propio, basado en C++.

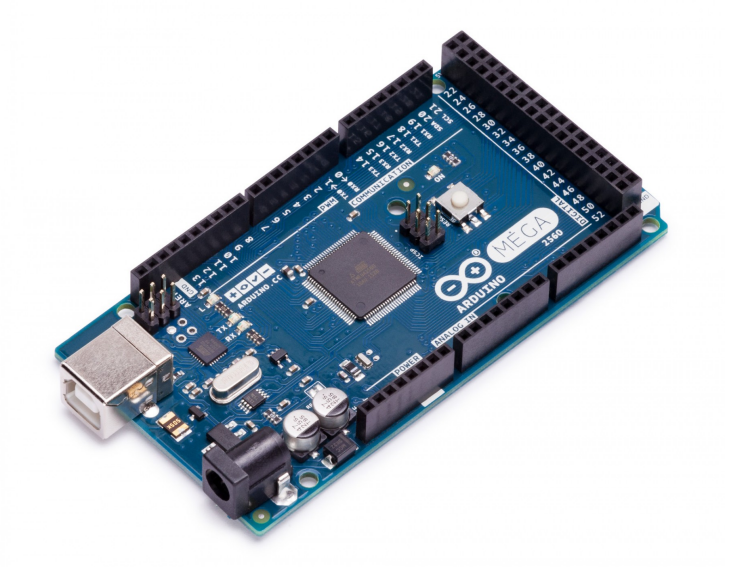


Figura 3.1.1.2 – Arduino Mega 2560 r3

3.1.2. Raspberry Pi 3B

En el año 2016 Raspberry sacó el modelo 3B de su serie de ordenadores de bajo coste dada a conocer en febrero del año 2012. Esta placa es, básicamente un ordenador con todos los componentes básicos que necesita para funcionar; procesador, procesador gráfico, memoria

RAM, conexión RJ45, 4 puertos USB 2.0, HDMI y compatibilidad almacenamiento externo en forma de micro SD. Además, implementa una serie de entradas y salidas de propósito general (Analógicas, Digitales, I2C...) que la hacen interesante para muchos proyectos en el ámbito de la electrónica. La Raspberry Pi ejecuta un sistema operativo instalado en una tarjeta micro SD. Existen una serie de sistemas operativos con diferentes aplicaciones en el ámbito multimedia, IoT y de propósito general, el mas extendido es Raspbian, un sistema operativo con núcleo Linux y una serie de herramientas instaladas que permiten trabajar con las entradas y salidas.

Figura 3.1.2.1 – Raspberry Pi 3B

3.1.3. Beaglebone Black (Rev. C)

La Beaglebone Black es un sistema embebido producido por Texas Instruments, lanzada su primera versión el 23 de abril del año 2013, y actualizada en repetidas revisiones hasta la actual, la Rev C, que cuenta como novedades respecto a su original predecesora una memoria eMMC de 4GB, en lugar de 2, lo que permitió a los desarrolladores entregarla con un sistema operativo basado en Linux preinstalado.

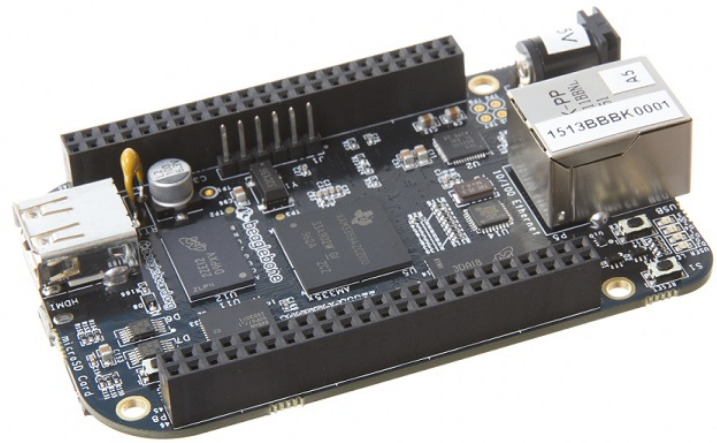


Figura 3.1.3.1 – Beaglebone Black Rev. C

Como características generales a nivel de hardware de la Beaglebone Black (en adelante, Beaglebone), tenemos que su núcleo es un microprocesador AM335x que funciona a 1GHz y está basado en el procesador ARM® Cortex-A8. También cuenta con 512MB de memoria RAM DDR3, una memoria no volátil (eMMC), en la que se puede instalar el sistema operativo (Debian por defecto), también tiene una ranura para tarjetas micro SD, un puerto USB 2.0 y un conector micro HDMI, aparte de los pines de entradas y salidas tanto analógicas como digitales.

A nivel de software, la Beaglebone soporta, de manera oficial, Angstrom Linux, Ubuntu y Android. Angstrom Linux es la distribución que viene instalada por defecto, y aparte de todas las ventajas que puede otorgarnos por ser una distribución Linux, cuenta con una herramienta realmente útil que nos facilitará la programación de la placa; se trata de Cloud9, un entorno de desarrollo con soporte para navegadores web que nos permite programar la Beaglebone desde el navegador de nuestra elección, utilizando para ello una conexión de tipo SSH a través del puerto USB de nuestro ordenador. Este software soporta una gran variedad de lenguajes de programación, y nos permite ejecutar los programas en el procesador la propia placa, a diferencia de Matlab, que utiliza la Beaglebone como una tarjeta de adquisición de datos, leyendo las entradas y actuando sobre las salidas.

3.2. BQ Hephestos Prusa i3

La Escuela Universitaria Politécnica de la Universidad de A Coruña dispone de una impresora BQ Hephestos Prusa i3, esta impresora, desarrollada por la compañía Española BQ y lanzada en el año 2013, cuenta con una superficie de impresión de 215mmx210mmx180mm, esto lo debemos tener en cuenta a la hora de diseñar las piezas de nuestro sistema, ya que no pueden superar estas dimensiones. Además, el filamento disponible es Acido Poliláctico (PLA), un polímero biodegradable obtenido a partir de almidón de maíz, yuca o caña de azúcar.

Figura 3.2.0.1 – BQ Hephestos Prusa i3

Se utilizará el software Cura para convertir los diseños 3D a un formato de archivo compatible con la BQ Hephestos Prusa i3. *Ver anexo 11.3*

3.3. Algoritmo de control PID

Como se menciona en el libro *Feedback Systems: An Introduction for Scientists and Engineers* [4], el control puede tener un amplio abanico de significados en función de quien utilice el término, pero en lo que a ingeniería respecta, el control es el uso de algoritmos y bucles de realimentación para conseguir que un sistema físico alcance el estado deseado.

En la actualidad, existen numerosos métodos mediante los que alcanzar el objetivo arriba descrito, uno de ellos, y el mas extendido por su simplicidad de implementación (que no de ajuste), por su versatilidad y por su correcto funcionamiento en sistemas SISO, es el algoritmo PID.

El algoritmo PID se basa en la suma de las acciones de los reguladores que lo forman, estos son el proporcional, el integral y el derivativo. Cada uno de ellos tiene un efecto muy concreto en la salida del sistema, y se pretende que la suma de dichos efectos haga a este alcanzar la estabilidad.

La ecuación 3.3.0.1 es la formulación del PID analógico.

$$u(t) = K_p e(t) + \frac{Kp}{Ti}\int_0^t e(\tau)d\tau + K_p T_d \frac{de(t)}{dt} \tag{3.3.0.1}$$

Esta formulación debe ser discretizada con el objetivo de implementarla en nuestro sistema embebido, en los siguientes puntos se explica cada uno de los términos del PID y la discretización de estos.

- Proporcional

  Es la parte del algoritmo producto de la constante proporcional por el error instantáneo. El error instantáneo es la diferencia entre el valor de consigna y la señal de salida; se pretende, con esta parte, que el error llegue a cero, pero utilizando únicamente el proporcional, siempre habrá un error de posición, además de sobreoscilaciones.

$$K_p E(S) \tag{3.3.0.2}$$

- Integral

  Tiene como propósito eliminar el error en estado estacionario provocado por perturbaciones externas y que no puede ser corregido por el proporcional, para ello integra la desviación entre la variable y la consigna y la multiplica por la constante integral. El efecto de esta acción se acumula, de forma que en caso de que la salida se sature, puede producirse un fallo en el control. La Implementación de un anti-windup solucionaría este problema

$$\frac{K_I}{S} E(S) \tag{3.3.0.3}$$

- Derivativo El control derivativo proporciona una respuesta a la derivada del error, es

decir, la velocidad de cambio de este último. Podría considerarse como un elemento de predicción dentro del PID que incrementa la estabilidad del sistema.

$$[K_D S]E(S) \tag{3.3.0.4}$$

El algoritmo PID que vamos a implementar debe ser discreto, ya que el controlador es digital. Por ello, se programará el algoritmo basado en la siguiente ecuación: 3.3.0.5

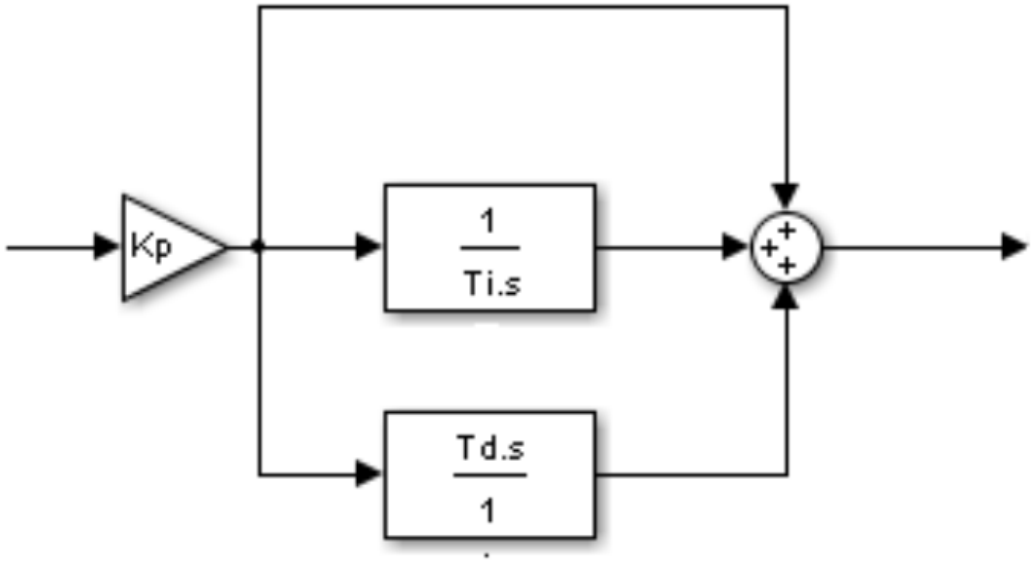


Figura 3.3.0.1 – PID estándar

$$U(t) = K_p[e(t) + \frac{T_c}{T_I}\sum_{i=1}^{t} e(i) + \frac{T_d}{T_c}[e(t) - e(t-1)]] \tag{3.3.0.5}$$

4 Normas y referencias

4.1. Disposiciones legales y normas aplicadas

Se aplicará la normativa de la Escuela Universitaria Politécnica en lo que respecta a la presentación de trabajos de fin de grado.

4.2. Bibliografía

[1] HIMPE, VINCENT; *Mastering the I2C bus*, 1ª ed. Susteren, Elektor Electronics Publishing, 2011.

[2] ALONSO, MARCELO; *Física*, Argentina; España. Addison-Wesley Iberoamericana, 1995.

[3] OGATA, KATSUHIKO; *Sistemas de control en tiempo discreto*,2ª ed. México, Prentice-Hall, 1996.

[4] JOHAN ÅSTRÖM, KARL; *Feedback Systems: An Introduction for Scientists and Engineers*,1ª ed. Princeton University Press, 2008.

4.3. Software Utilizado

- **Matlab R2017a:** Obtención de gráficas a partir de datos extraídos de los sensores
- **Siemens NX 11.0:** Diseño de las piezas que conforman el sistema objeto.
- **Cloud9:** Programación de los algoritmos.
- **TeXnicCenter:** Redacción de la documentación.
- **MiKTeX:** Compilador utilizado por TeXnicCenter.
- **CURA:** Conversión de las piezas 3D generadas en NX al formato de la impresora 3D.
- **Putty:** Alternativa a Cloud9 para conectarse via SSH a la Beaglebone Black
- **Win32Imager:** Grabación de imagen ISO en tarjeta Micro SD.
- **7zip:** Descompresión de imagen ISO.

4.4. Otras referencias

[I] *Imagenes de Debian*, [11/07/2017]. Disponible en: https://beagleboard.org/latest-images

[II] *7zip*, [11/07/2017]. Disponible en: http://www.7-zip.org/

[III] *Win32Imager*, [11/07/2017]. Disponible en: https://sourceforge.net/projects/win32diskimager/

[IV] *Putty*, [11/07/2017]. Disponible en: http://www.putty.org/

[V] *Instalación de la librería I2C*, [11/07/2017]. Disponible en: https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/installation-on-ubuntu

[VI] *Datasheet y Mapa de Registros*, [11/07/2017]. Disponible en: https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/

[VII] *Tilt Sensing Using a Three-Axis Accelerometer*, [30/08/2017]. Disponible en: http://www.nxp.com/docs/en/application-note/AN3461.pdf

[VIII] *Setting the I2C bitrate on Beaglebone Black*, [11/07/2017]. Disponible en: https://groups.google.com/forum/#!msg/beagleboard/vbuM-4oShS8/yk7V21qowIgJ

[IX] *The Practical Guide of Multicopters*, [11/07/2017]. Disponible en: http://thepracticalguideofmulticopters.com/index.php?id_cms=14&controller=cms

[IX] *Control PID*, [11/07/2017]. Disponible en: http://control-pid.wikispaces.com/#CONTROL%20PID,%20METODOLOG%C3%8DA%20Y%20APLICACIONES-Reglas%20heur%C3%ADsticas%20de%20ajuste:

5 Definiciones y abreviaturas

- **BBB:** Beaglebone Black.
- **PRU:** Programable Real-Time Unit (Unidad programable que funciona en tiempo real).
- **SISO:** Single Input Single Output (Una entrada, una salida).
- **Cabeceo:** Rotación respecto al eje central paralelo a propulsor-propulsor, en el documento, puede utilizarse Roll, que es su forma inglesa.
- **Alabeo:** Rotación respecto al eje contrapeso-propulsores, en el documento, puede utilizarse Pitch, que es su forma inglesa.
- **Guiñada:** Rotación intrínseca alrededor del eje vertical perpendicular al sistema, en el documento, puede utilizarse Yaw, que es su forma inglesa.
- **MEMS:** Sistema Micro-Eléctrico Mecánico.
- **ESC:** Electronic Speed Controller (Controlador Electrónico de Velocidad).
- **I2C:** Inter-Integrated Circuit (Bus de datos serie).
- **PID:** Proportional Integral Derivative (Proporcional Integral Derivativo), hace referencia al conocido mecanismo de control realimentado.
- **PWM:** Pulse-Width Modulation (Modulación por ancho de pulso).
- **PPM:** Pulse-Position Modulation (Modulación por posición de pulso).
- **RPM:** Revoluciones Por Minuto.

6 Requisitos de diseño

El primer objetivo de este proyecto será desarrollar un sistema formado por dos motores brushless y que cuente con tres grados de libertad, de forma que se pueda este mover libremente en cualquier dirección del espacio.

Será necesario hacer contar a este sistema con una unidad de medición inercial, de forma que podamos conocer la posición espacial en cualquier instante de tiempo.

También deberemos implementar y ajustar un algoritmo de control PID en un sistema embebido, que tome como entrada los datos de la unidad de medición inercial y actúe en consecuencia sobre la salida, es decir, sobre la velocidad de los motores brushless.

El sistema en su conjunto debe permanecer indefinidamente en una posición estable.

7 Análisis de las soluciones

7.1. Selección del sistema embebido

Cualquiera de las tres placas sirve para llevar a cabo el proyecto, todas ellas soportan el protocolo I2C y tienen salidas PWM, además, existen antecedentes de proyectos similares con cada una de ellas. Al comenzar este proyecto, estaba interesado en el lenguaje de programación Python, y tanto Raspberry Pi como Beaglebone disponían de entornos de desarrollo en dicho lenguaje, pero la Raspberry esta mas orientada hacia el mundo multimedia, y no dispone de una gama tan amplia de entradas y salidas, como podemos ver en las figuras 7.1.0.1 y 7.1.0.2. Además, este proyecto está orientado al desarrollo de un multicóptero, y la Beaglebone cuenta con una PRU que es necesaria si queremos asegurarnos de llevar a cabo un control manual del vehículo en tiempo real, ya que no existe garantía alguna de que el procesador sobre el que se ejecuta el código de estabilización y control se ejecute de este modo, dado que tanto la Beaglebone Black y la Raspberry Pi 3b ejecutan sendos sistemas operativos sobre sus respectivos procesadores. Esto último inclinó la balanza en favor de la Beaglebone Black, de forma que será la placa que se utilizará en este proyecto.

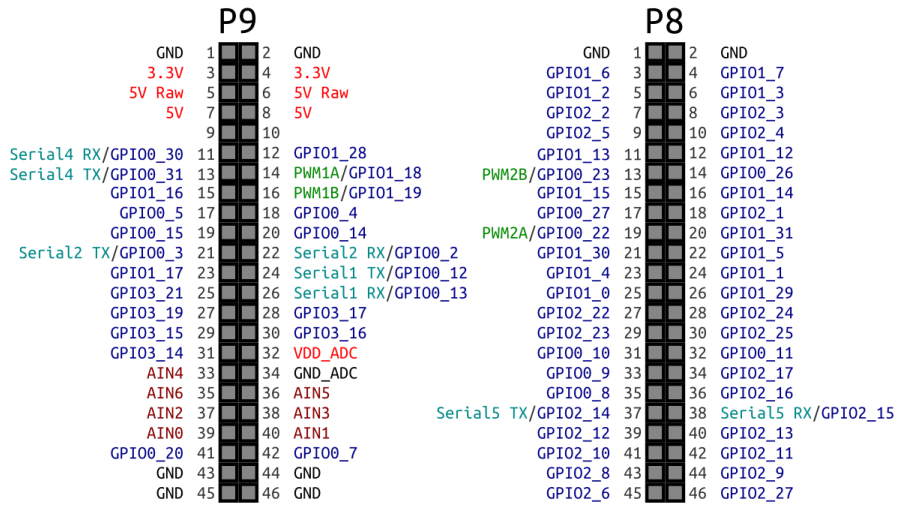


Figura 7.1.0.1 – Pinout de la Beaglebone Black Rev. C

Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figura 7.1.0.2 – Pinout de la Raspberry Pi 3B

7.2. Selección del sensor

Como ya sabemos, el objetivo de este proyecto es estabilizar un sistema con tres grados de libertad, y una de las condiciones indispensables para ello es conocer la posición exacta en cada instante de tiempo del sistema. Para ello, podríamos simplemente usar 3 encoders angulares, que nos devolverían con total precisión la posición angular de cada uno de los 3 ejes. Pero este proyecto será el precedente de un futuro proyecto que pretende estabilizar un multicóptero, el cual no tendrá contacto alguno con el suelo. Teniendo esto en cuenta, la única forma que tenemos de cuantificar el Cabeceo, el Alabeo y la Guiñada es utilizar un Acelerómetro, un Giroscopio y un Magnetómetro.

Inicialmente, podríamos pensar que únicamente necesitaríamos un acelerómetro para calcular el cabeceo y el alabeo, obteniendo dichos valores mediante trigonometría, pero como se demostrará mas adelante, los valores obtenidos únicamente mediante las mediciones del acelerómetro no son válidos para este proyecto.

7.2.1. Acelerómetro

Por definición, un acelerómetro es un instrumento utilizado para medir aceleraciones, pueden medir tanto la aceleración de coordenadas, que es el cambio en la magnitud de la velocidad del objeto en el espacio, o la aceleración del campo gravitatorio de la tierra, en caso de

este permanecer fijo en el suelo.

Existen muchos tipos de acelerómetros, algunos de un tamaño que no tendría sentido en el proyecto, por ello, me centraré en los acelerómetros MEMS (*micro electro-mechanical systems*), este tipo de acelerómetros están formados por un elemento estructural rígido al que está acoplado una pequeña masa de prueba. Bajo la influencia de aceleraciones externas, la masa de prueba se desplaza de su posición de reposo, siendo medido este desplazamiento, generalmente mediante el cambio de capacidad de un condensador, para obtener el valor de la aceleración. En la figura 7.2.1.1 podemos observar el interior de un acelerómetro MEMS basado en el cambio de capacidad de un condensador.

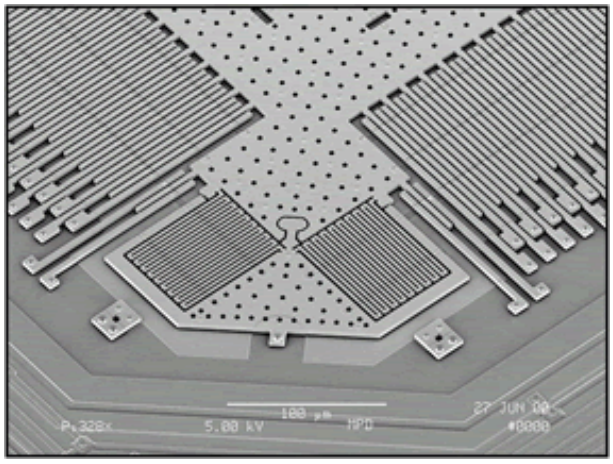


Figura 7.2.1.1 – Vista microscópica de un acelerómetro MEMS

7.2.2. Giroscopio

Tradicionalmente, la idea de giroscopio se corresponde con un dispositivo formado por una rueda giratoria, montada sobre un eje que puede cambiar libremente de dirección. La rueda gira rápidamente alrededor del eje principal, de forma que, si movemos el giroscopio, la rueda permanecerá fija en su posición inicial, mientras que el eje se ha movido.

Los giroscopios MEMS (7.2.2.1) tienen un funcionamiento muy diferente, estos miden la velocidad angular a la que está girando un eje determinado, utilizando para ello la fuerza de Coriolis, que hace vibrar dos masas que forman un condensador de capacidad variable; esta capacidad es la que se medirá para obtener el valor de velocidad angular.

7.2.3. Magnetómetro

Un magnetómetro cuantifica la fuerza del campo magnético que actúa sobre el. A pesar de que es un dispositivo sensible al campo magnético generado por cualquier objeto, se suele utilizar para conocer la guiñada de un sistema; esto sería la desviación en grados con respecto al polo norte magnético.

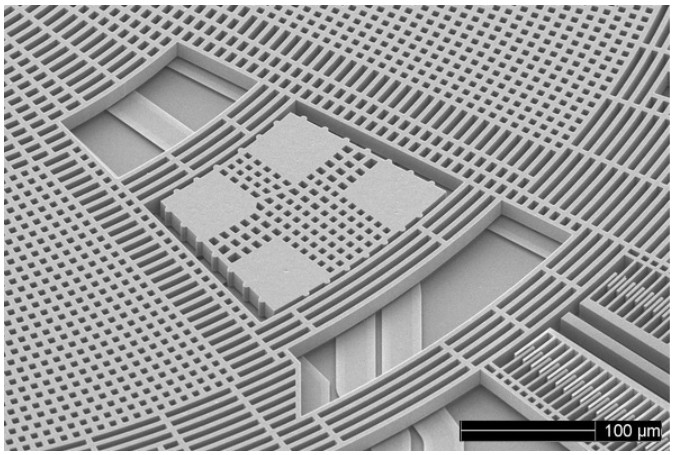


Figura 7.2.2.1 – Vista microscópica de un giroscopio MEMS

7.2.4. IMU

Una IMU (*Inertial Measurement Unit*) es un sistema generalmente formado por un giroscopio y un acelerómetro, que puede en ocasiones llevar integrado un magnetómetro, y cuya finalidad es entregar al usuario datos precisos de su posición espacial. Si comparamos las IMU comerciales con los giroscopios y magnetómetros MEMS, no difieren mucho en cuanto a características técnicas, ni siquiera en cuanto a precio, pudiendo encontrar una IMU que consta de acelerómetro, giroscopio y magnetómetro por un precio inferior a sus partes por separado, además de, al ser un único dispositivo, y no 3, ocupar mucho menos espacio.

Teniendo todo lo anterior en cuenta, lo lógico es decantarse por una IMU. A nivel comercial, existen muchas IMUs de bajo precio creadas para el mundo *maker*, por lo que lo ideal es conseguir una de ellas. Invensense desarrolló un chip llamado MPU-9250 que cumple con todos nuestros requisitos, cuenta con un Acelerómetro capacitivo, un Giroscopio, también capacitivo y un sensor de efecto Hall que hace de magnetómetro; todos ellos de 3 ejes. Muchos fabricantes han integrado este chip en un circuito impreso con los componentes discretos necesarios para su correcto funcionamiento, por lo que será tomada esta solución.

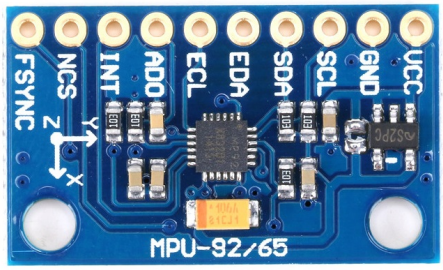


Figura 7.2.4.1 – MPU9250

7.3. La comunicación I2C

I2C es un protocolo de comunicación desarrollado por Philips Semiconductors con el objetivo de controlar circuitos integrados de televisión mediante un bus simple (dos hilos). Está orientado a transferencias de datos de 8 bits, y las velocidades que se pueden alcanzar van desde los 100kbit/s en el modo estandar hasta los 3.4Mbit/s en el modo de alta velocidad. Existen modos intermedios, de 400kbit/s y 1Mbit/s. En lo que respecta al bus, se trata de un sistema síncrono con un sistema de comunicación Maestro-Esclavo, que consiste en dos hilos que transportan la información en forma digital. Estos son SDA y SCL. SDA transporta los datos en serie, mediante una comunicación Half duplex y SCL es el reloj. Por supuesto, la masa debe ser común entre el maestro y el esclavo. Los dispositivos dentro del bus deben tener una dirección única, el maestro comienza la transmisión generando una señal de reloj, momento en el que los dispositivos direccionados comienzan a actuar como esclavos.

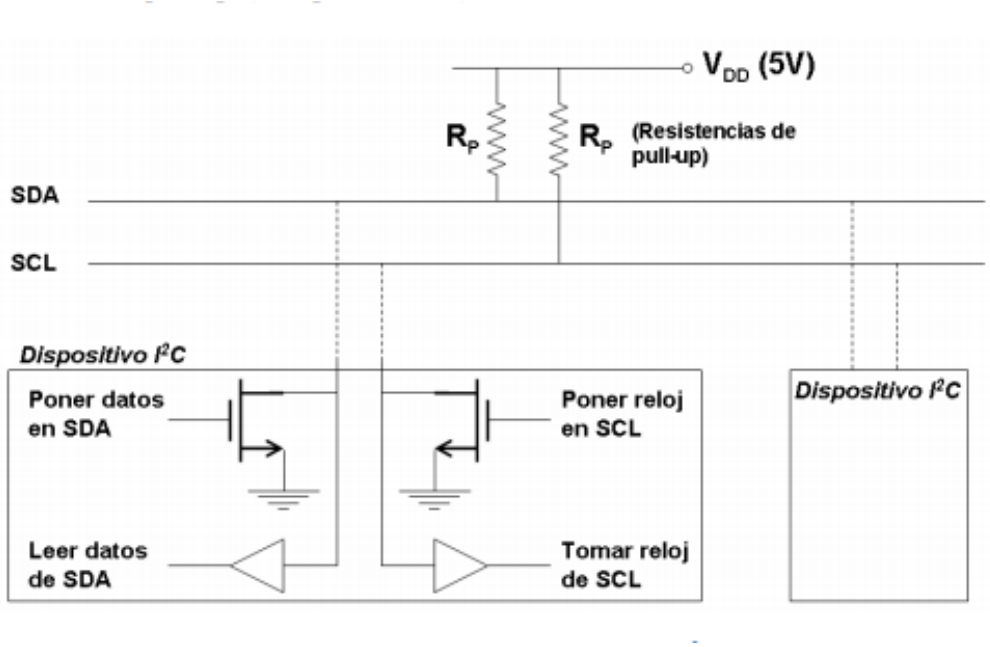


Figura 7.3.0.1 – Conexión de dispositivos al bus I2C

7.3.1. Transferencia de datos

Cuando se da una condición START (Figura 7.3.1.1), los datos se comienzan a transmitir, siempre en serie y en formato Byte con el bit mas significativo en primer lugar. Al final de la transmisión, se emite un bit extra cuyo objetivo es indicar al maestro que esta se ha llevado a cabo de forma correcta; en caso de no producirse esto último, el maestro emitirá o bien una condición de STOP o repetirá el START para volver a transferir el dato

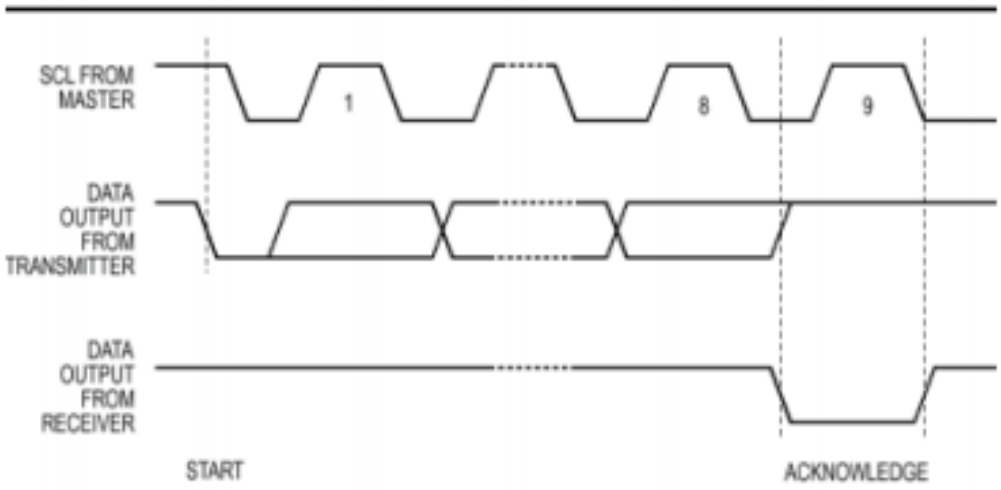


Figura 7.3.1.1 – Transferencia de un Byte

7.4. El motor brushless

El motor brushless es un motor eléctrico que, como su nombre indica, carece de escobillas. Los motores eléctricos tradicionales funcionan mediante escobillas, como podemos apreciar en la figura 7.4.0.1, en la parte exterior se colocan los imanes permanentes, y en la interior el rotor, formado por un núcleo de material ferromagnético bobinado. Cada vez que el motor gira 180 grados, se invierte la polaridad del rotor, de forma que este se ve repelido por el imán permanente adyacente. Esta secuencia se repite indefinidamente, manteniéndose así un movimiento constante.

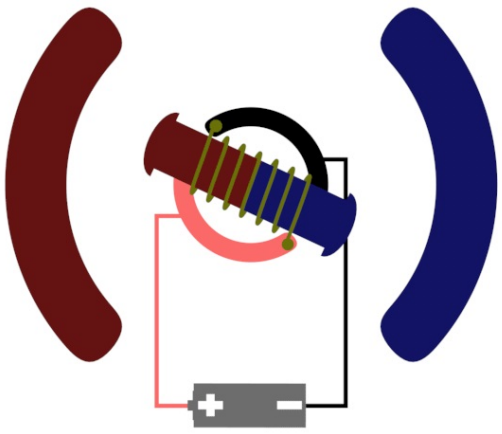


Figura 7.4.0.1 – Motor Eléctrico de Escobillas

Aunque realmente, los motores con únicamente dos polos como el de la imagen 7.4.0.1 no son utilizados, ya que se necesita una cierta inercia de giro para arrancar. Por ello, se suele incorporar una tercera escobilla, figura 7.4.0.2

El motor brushless prescinde de las escobillas, en su lugar, utiliza devanados que se activan ordenada y secuencialmente gracias a un control electrónico. Esto supone una serie de

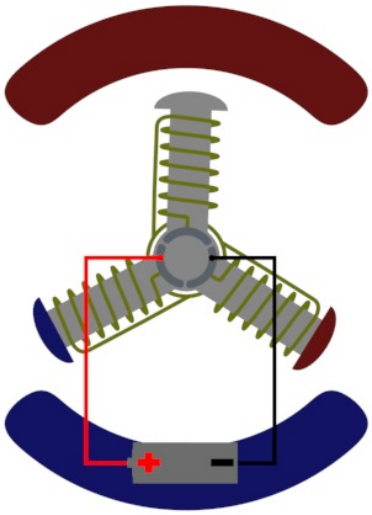


Figura 7.4.0.2 – Motor Eléctrico de tres polos

ventajas del motor brushless sobre el motor de escobillas tradicional:

- Mayor vida útil debido a que no existe en desgaste de escobillas
- Menos ruido
- Mejora de la refrigeración
- Desaparición de chispas y ruido electromagnético

En la figura 7.4.0.3 se ve la construcción de un motor brushless básico, que cuenta con 3 electroimanes y dos imanes permanentes, esto se expresaría como (3N2P). La corriente circula alternativamente en la siguiente secuencia (entiéndase el electroimán a la izquierda de la flecha como polo positivo):

1. Corriente A $\rightarrow$ B
2. Corriente C $\rightarrow$ B
3. Corriente C $\rightarrow$ A
4. Corriente B $\rightarrow$ A
5. Corriente B $\rightarrow$ C
6. Corriente A $\rightarrow$ C

Sin embargo, esta no es la configuración ideal de un motor Brushless teniendo en cuenta el tamaño de la Beaglebone Black y la orientación de este proyecto, dado que este tipo de motores alcanzan un alto número de revoluciones, pero solo son capaces de mover pequeñas hélices debido a su bajo par motor, lo que se traduce muy poco empuje. Por ello, debemos

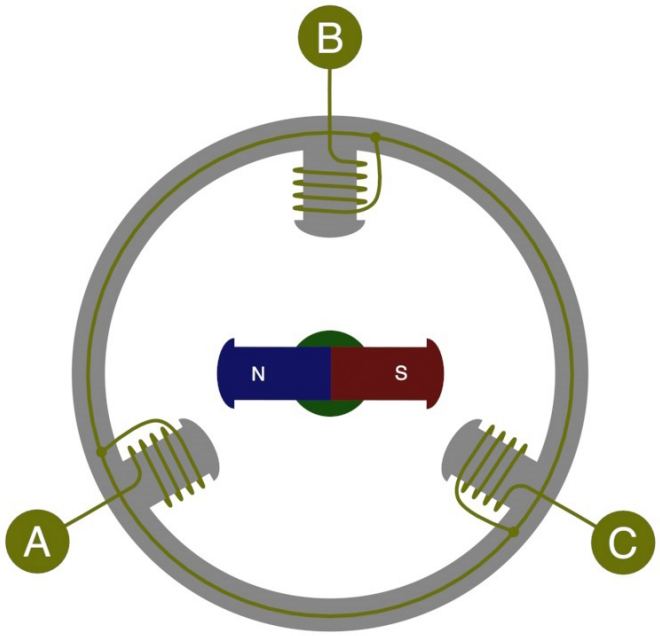


Figura 7.4.0.3 – Motor Brushless Inrunner

utilizar un motor brushless outrunner. La diferencia a nivel de contrucción entre ambos tipos de motor, como se puede inferir de sus nombres, radica en que los inrunner tienen los electroimanes en el exterior, y lo que gira es un eje de imanes permanentes; mientras que los outrunner tienen los electroimanes en el interior, y lo que gira es una campana exterior en la que se encuentran los imanes permanentes.

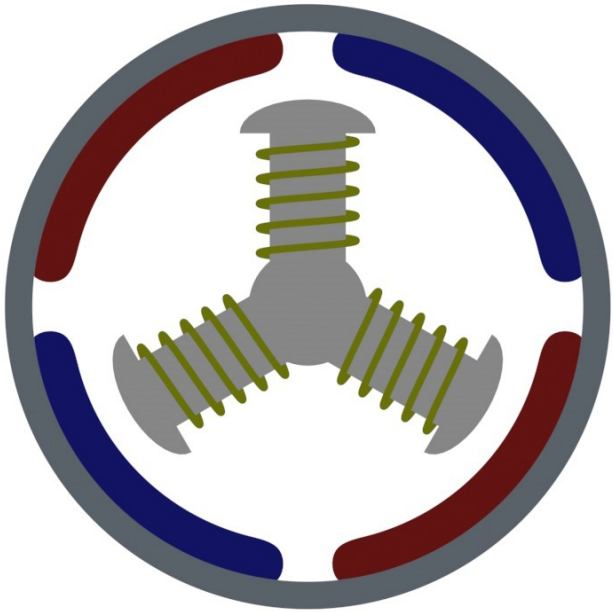


Figura 7.4.0.4 – Motor Brushless Outrunner

Para seleccionar el motor adecuado, serán analizados sus parámetro técnicos básicos, estos son la potencia máxima y la relación entre RPM y tensión de alimentación, ya que no existe una amplia variedad de motores cuyas especificaciones completas sean dadas por el fabricante, y estas son las mas habituales.

7.5. El ESC

El ESC (Electronic Speed Controller) es un dispositivo electrónico que puede controlar la velocidad y dirección de giro de un motor eléctrico. Básicamente, lo que hacen es generar una señal trifásica gracias al *switcheo* de los transistores FET que lo forman, en función de la anchura de pulso que llegue al ESC, que suele variar entre 1ms y 2ms para una frecuencia de 50Hz, los transistores se activan en una misma secuencia a mayor o menor frecuencia, incrementando de este modo la velocidad de cambio de polos dentro del motor brushless y por ende la velocidad y el consumo. Cabe mencionar que la activación y desactivación de los transistores es controlada por un microcontrolador integrado en el propio ESC.

En la imagen 7.5.0.1 se puede observar el desfase de 120° de las señales generadas por el ESC, que causan la inversión de la polaridad en los polos del motor. En la imagen 7.5.0.2, la matriz de transistores que permite generar dichas señales

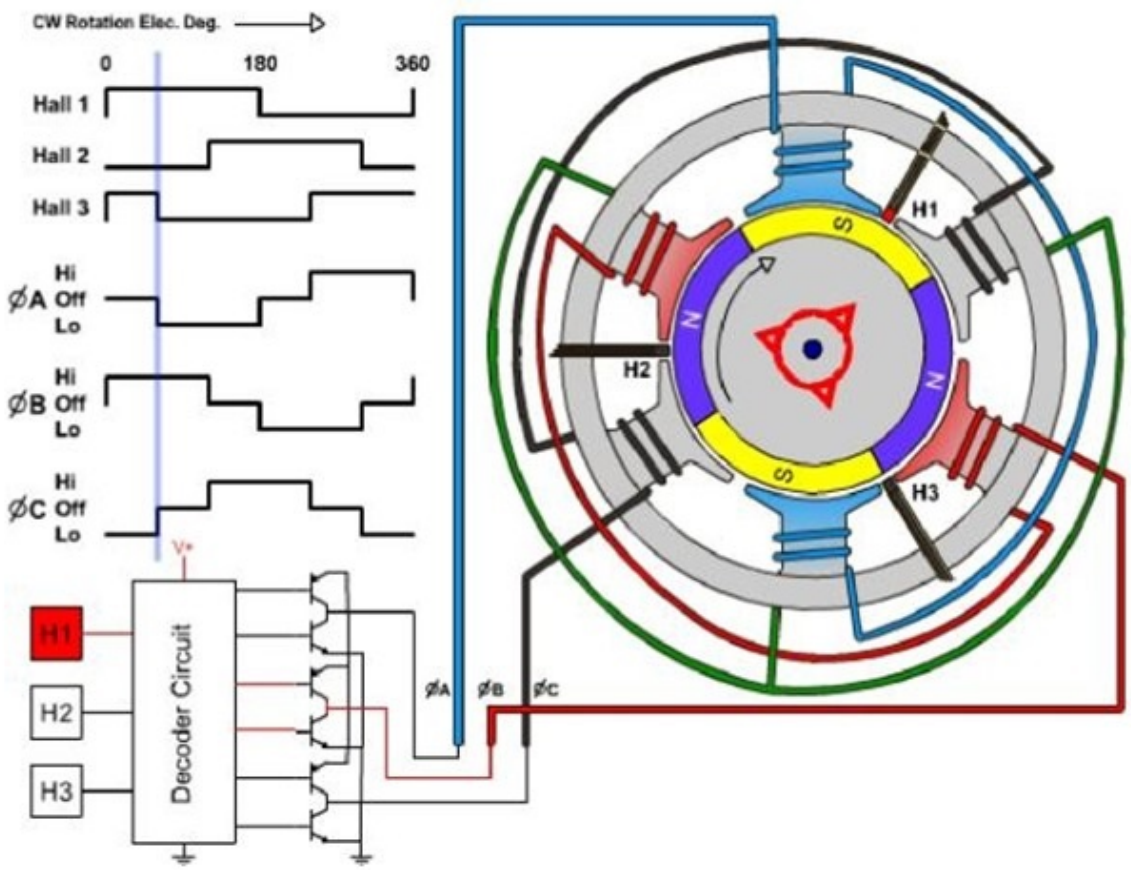


Figura 7.5.0.1 – Señales generadas por un ESC

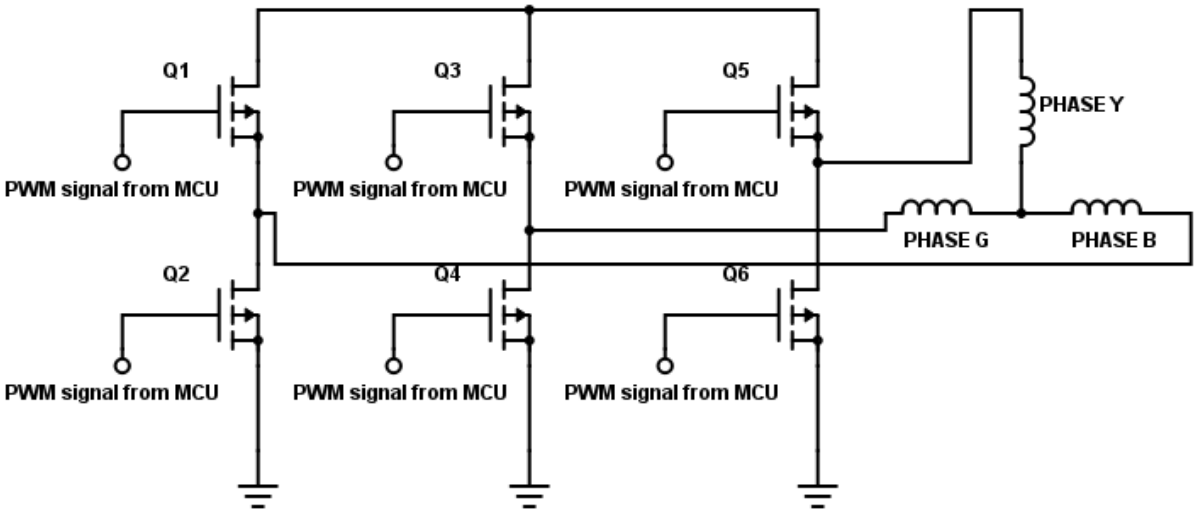


Figura 7.5.0.2 – Transistores en un ESC

8 Resultados finales

8.1. El sistema objeto

Imagen 8.1.0.1, representa la apariencia final del sistema objeto.

Figura 8.1.0.1 – El Sistema Objeto

8.1.1. Rótula

La rótula es, estructuralmente, el núcleo del proyecto. Permite a este moverse alrededor de tres ejes, aunque al final de este proyecto solo dos de ellos sean utilizados (Esto último se explica en la sección 8.3.3).

Se compone, principalmente de 4 piezas impresas en 3D a medida, 6 rodamientos y 3 varillas que atraviesan estos últimos; los rodamientos encajan en los agujeros a la perfección, pero es recomendable aplicar algún tipo de adhesivo, preferiblemente cianocrilato. Podemos ver, en la imagen 8.1.1.1 el ensamblaje de todos estos elementos, a falta de un motor reciclado que va anclado a la propia rótula y a la base y que funciona como eje Z. Respecto a este último, se han diseñado los objetos que se describen en el Plano N°7 y el Plano N°8 a medida para

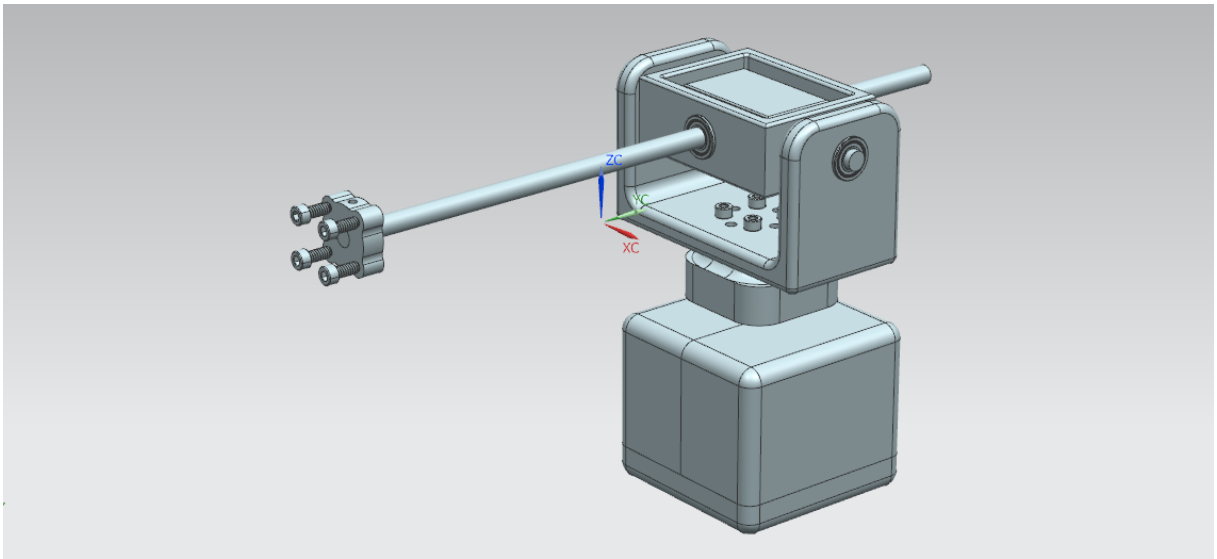


Figura 8.1.1.1 – Ensamble de la Rótula

ser encajado en su interior. Además del objeto descrito por el Plano N°6 como adaptador entre la rótula y el eje de dicho motor.

8.1.2. Sistema de Propulsión

Se compone de un ensamble como el que vemos en la figura 8.1.2.1 y dos motores brushless 8.1.2.2 con sus propulsores 8.1.2.3. Las diferentes piezas del ensamble se unen con un adhesivo de cianocrilato, se montan las hélices en los motores, teniendo en cuenta el apriete de las tuercas (sentido horario y antihorario). Por último se atornillan los motores a la base del ensamble.

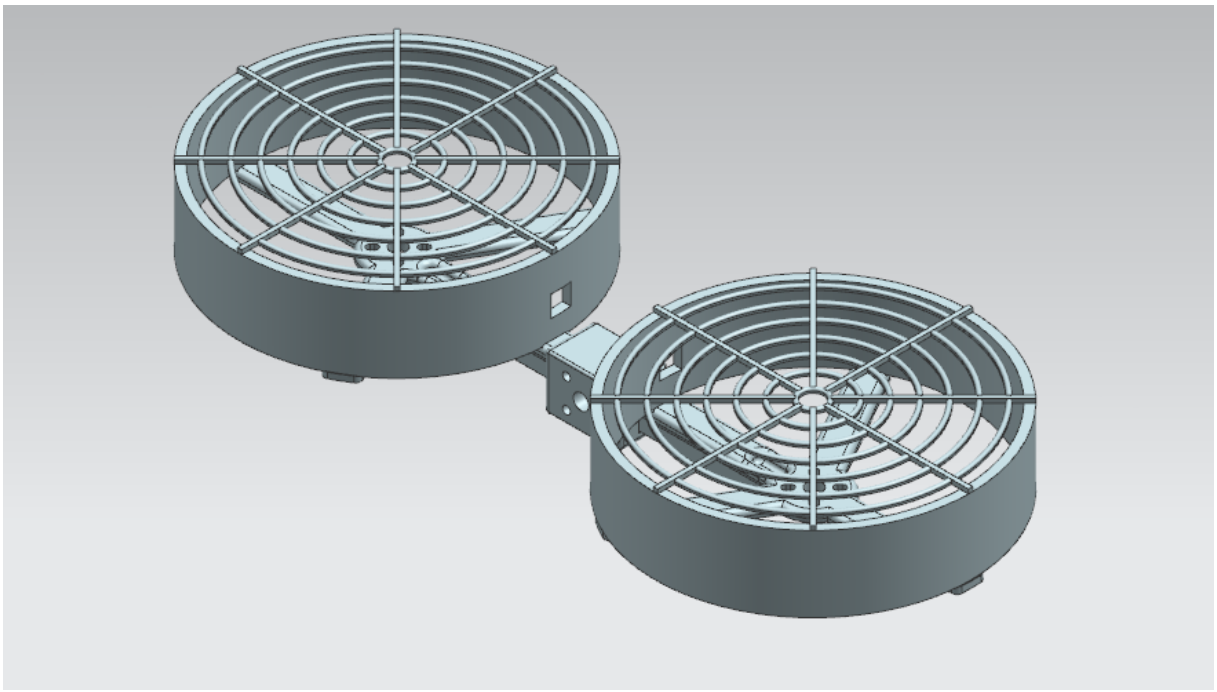


Figura 8.1.2.1 – Ensamble del sistema de propulsión

Figura 8.1.2.2 – Motores Brushless Utilizados

Figura 8.1.2.3 – Hélices 5030

El sistema de propulsión se unirá a la varilla que atraviesa la rótula con un acople de tornillo de 1/4 de pulgada.

8.1.3. Electrónica y Alimentación

El diagrama electrónico de la figura 8.1.3.1 indica las diferentes conexiones del sistema, tanto de la parte de potencia como de la de señal.

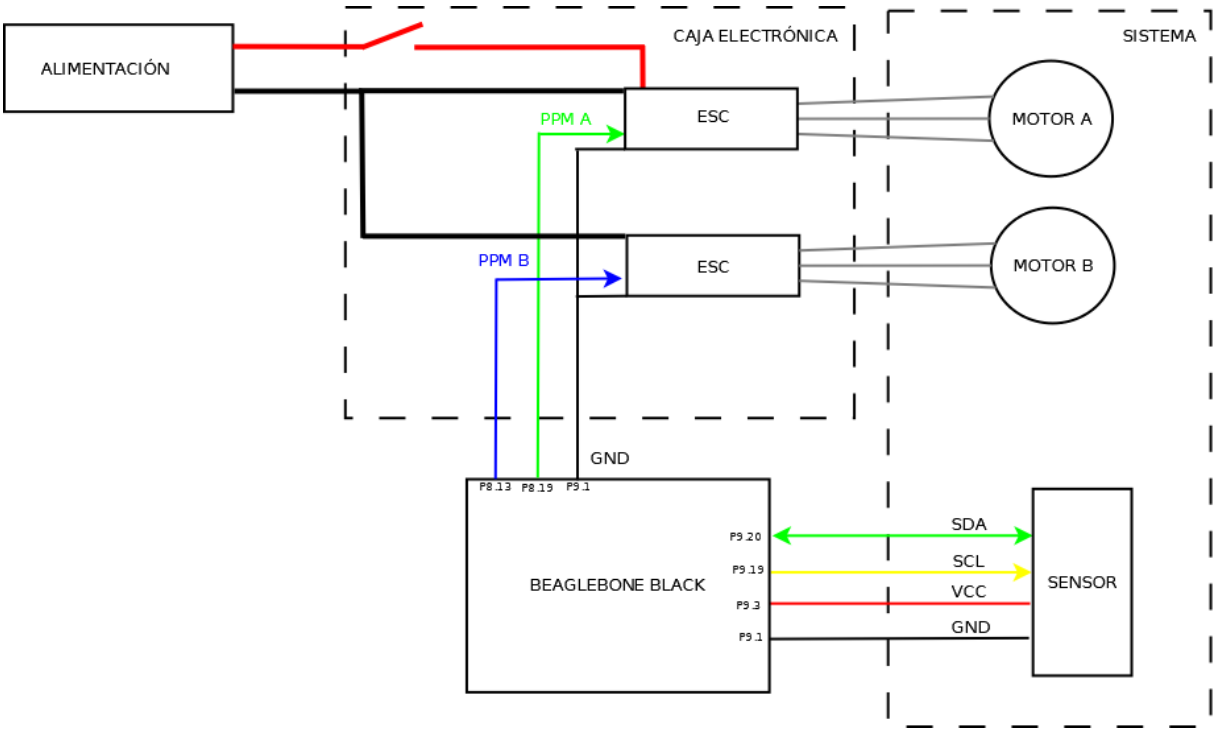


Figura 8.1.3.1 – Diagrama Electrónico

8.1.3.1. Batería

Para sacar el máximo partido de las especificaciones de los motores y ESC, se requiere de una batería capaz de suministrar holgadamente la tensión y corriente que estas necesitan, por esto, me he decantado por una batería de 4 celdas (4S) de 14.7V, 5000mAh de capacidad y una descarga de 25C a 35C, de marca Turnigy, que se muestra en la imagen 8.1.3.2 (un C es una unidad de capacidad, la batería tiene 5Ah, 25C quiere decir que la batería puede entregar $25 * 5 = 125A$ de forma continuada).

Figura 8.1.3.2 – Batería utilizada en el proyecto

8.1.3.2. ESC

Al igual que en el caso de los motores Brushless, existe una gran cantidad de ofertas de ESC en el mercado, por lo que resulta muy complicado decidirse por un producto concreto. Además, las hojas de características u especificaciones son prácticamente inexistentes y la única referencia que existe es la opinión de otros usuarios. Si se tiene esta en cuenta, los mejores ESC disponibles en el mercado, en la fecha de compra de los componentes, eran los ESC FVT Littlebee 30A, programados con un Firmware llamado BLHeli. Estos ESC, que podemos ver en la imagen 8.1.3.3 soportan una corriente de hasta 30A y baterías 4S. Determinados parámetros de estos reguladores de velocidad se pueden programar con un software y un adaptador especial (8.1.3.4), como se explica en uno de los anexos.

Figura 8.1.3.3 – ESC Littlebee

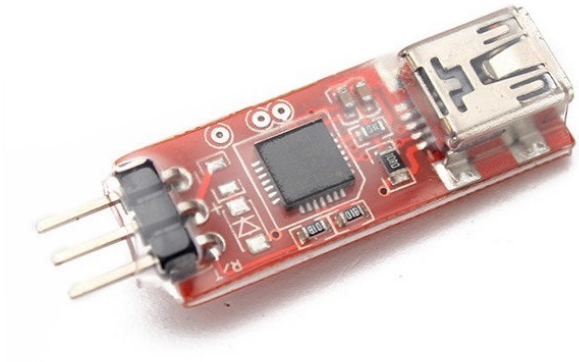


Figura 8.1.3.4 – Adaptador USB para programar los ESC

En lo que respecta a la conexión de estos a los motores, lo único que se debe tener en cuenta es que el cable central de salida del ESC debe ir conectado al cable central de entrada del motor Brushless, la conexión de los otros dos cables solo determina el sentido de giro del motor, que se puede modificar posteriormente por software. Esta conexión se realizará mediante una ficha, y no mediante soldadura, de manera provisional, para poder reutilizar en un futuro los componentes.

Otros componentes que forman la electrónica de este sistema son un cable adaptador para la batería (Imagen 8.1.3.5), dos conectores de 3.5mm (Imagen 8.1.3.6) y un indicador de voltaje, para no descargar la batería por debajo de un valor de seguridad, protegiendo así su vida útil (Imagen 8.1.3.7). También se ha utilizado un interruptor reciclado, de forma que en caso de emergencia, se pueden apagar los motores con rapidez y seguridad. Todos estos componentes, al igual que los ESC, se insertan dentro de una caja diseñada exclusivamente para ello, que se puede ver en el plano N°16.

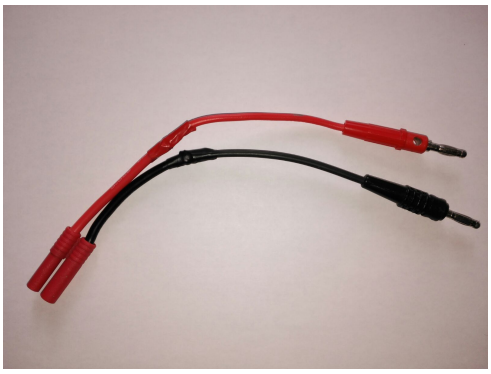


Figura 8.1.3.5 – Adaptador Batería a Conector Macho 4mm

Figura 8.1.3.6 – Conector Hembra 4mm

Figura 8.1.3.7 – Indicador de Tensión LED

La apariencia final es la que se muestra en la figura 8.1.3.8

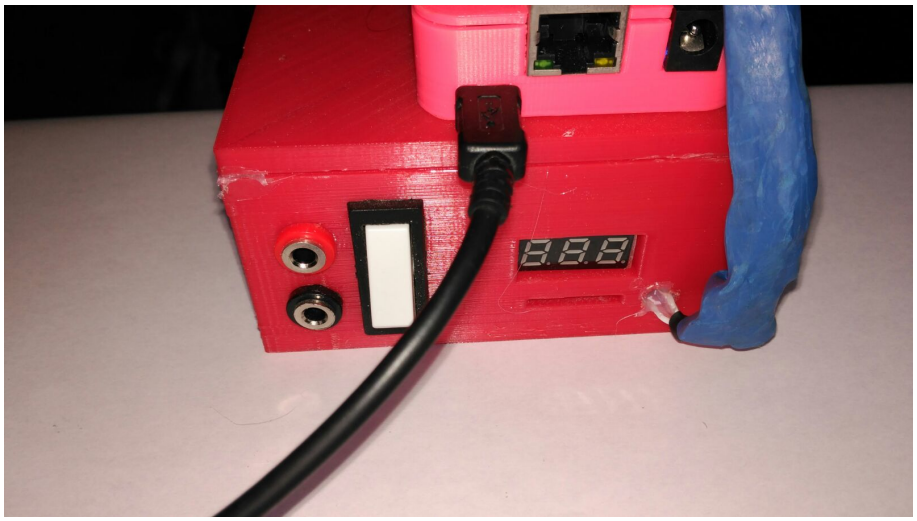


Figura 8.1.3.8 – Apariencia de la Caja

Podemos observar como van insertados los componentes, así como el cableado de datos (I2C y PWM) que va dentro del termorretráctil azul. En el interior, se encuentran las conexiones, que siguen el siguiente esquema 8.1.3.9:

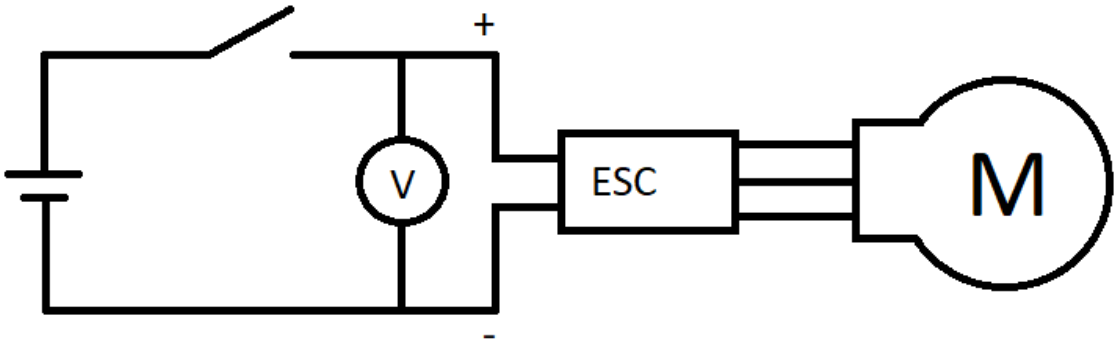


Figura 8.1.3.9 – Esquema Eléctrico

8.2. Lectura del bus I2C

Para leer los datos del sensor, hacemos uso del bus I2C. Este bus utiliza 2 cables para la transmisión de datos y otros 2 para la alimentación, la alimentación será suministrada por la BBB y los dos de datos (SPI y SDA), se conectarán a los pines 19 y 20 del P9 (Ver Imagen 7.1.0.1). El sensor del que hacemos uso soporta una velocidad de I2C de hasta 400KHz, pero por defecto, la Beaglebone está configurada para 100KHz, esto se debe de modificar siguiendo las especificaciones redactadas en [VIII]. Adafruit nos proporciona una serie de librerías que permiten interactuar, desde Python, con una serie de entradas y salidas de la Beaglebone. En la sección *Otras referencias* de la bibliografía de este documento, podemos encontrar los pasos que se deben seguir para instalar correctamente las librerías. A continuación, podemos ver el extracto del código cuyo objetivo es obtener los datos vía I2C. Debemos importar previamente las librerías que hemos instalado para que el programa funcione correctamente. Esto está debidamente comentado en el anexo *Listado de códigos de programación*.

Código 8.1: Lectura de los registros

```
# Funcion que lee un registro
#-------------------------------------------------------------------------------
def Read_I2C (MSB_Register, LSB_Register, Full_Scale_Value, MPU9250_AK8963) :
# Esta funcion tiene varias entradas, las dos primeras le indican que registro
# debe leer, la tercera, el valor a fondo de escala, que se utilizara para saber
# la escala en la que la IMU nos devuelve el valor, y la cuarta, que es un "bit"
# que especifica si se va a leer el acelerometro/giroscopio o el magnetometro.
    if MPU9250_AK8963 == 0 :
        High_Byte = I2C_MPU9250.readU8(MSB_Register)
        Low_Byte  = I2C_MPU9250.readU8(LSB_Register)
    else :
        High_Byte = I2C_AK8963.readU8(MSB_Register)
        Low_Byte  = I2C_AK8963.readU8(LSB_Register)
# Las siguientes lineas desplazan y encadenan el byte mas significativo y el
# menos significativo, de forma que nos queda un resultado de 16 bits, que en la
# tercera linea convertimos a un valor en unidades de aceleracion, velocidad
# angular o intensidad de campo magnetico
    High_Byte = High_Byte << 8
    Read_Value = High_Byte | Low_Byte
    Read_Value = (Read_Value/32768.0) * Full_Scale_Value
# Estas lineas de codigo aseguran que el valor devuelto sea positivo, y si no
# lo es, el signo se modifica, ya que el bit mas significativo es un bit de
# signo
    if Read_Value >= Full_Scale_Value  :
        Read_Value = Read_Value - (2 * Full_Scale_Value)

    return Read_Value
#-------------------------------------------------------------------------------
```

En la web de Invensense [VI], desarrollador del integrado MPU9250 podemos descargar tanto su datasheet como el mapa de registros. Este último es realmente importante, ya que debemos conocer en qué registros se encuentran almacenados los valores leídos por los diferentes sensores. Estos registros se han volcado en un archivo Python llamado *Registers.py* que podemos ver en el anexo *Listado de códigos de programación* con el objetivo de que su uso fuese mas sencillo, ya que en lugar de escribir un código en hexadecimal, escribimos el nombre con el que aparece el registro en el datasheet.

8.3. Acondicionamiento del sensor

La IMU de nuestro proyecto nos devuelve una serie de datos de aceleración, velocidad angular e intensidad de campo magnético que no son válidos para introducir directamente al algoritmo PID. Estos datos requieren de una serie de pasos de procesamiento, de forma que al final, obtengamos unos valores que reflejen fielmente la posición espacial, en coordenadas ortogonales, respecto a la superficie de la tierra y al Polo Norte magnético.

8.3.1. Estimación de Cabeceo y Alabeo

Con el objetivo de medir la orientación con respecto al campo gravitatorio terrestre, las ecuaciones que siguen, por convención, niegan el valor de salida de los 3 ejes del acelerómetro (figura 8.3.1.1, de forma que cualquiera de los últimos, en caso de estar alineado con el campo gravitatorio terrestre, nos de una salida de +1g.

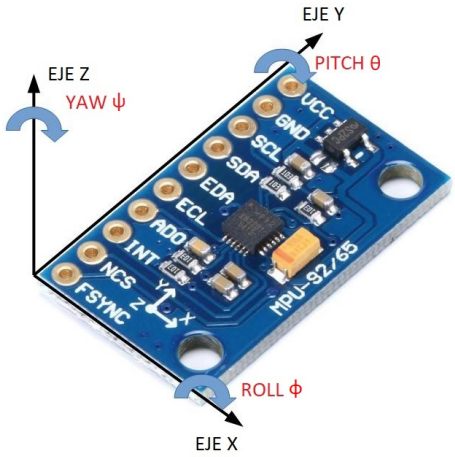


Figura 8.3.1.1 – Sistema de Coordenadas y Ejes de Rotación

Si tenemos esto en cuenta, el acelerómetro integrado en nuestra MPU9250 nos dará una salida (G_p), sometido a una aceleración lineal a_r, con respecto al marco de referencia terrestre de:

$$G_p = \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = R(g - a_r) \tag{8.3.1.1}$$

Donde R es la matriz de rotación que describe la orientación relativa al sistema de coordenadas terrestre.

Pero para llevar los cálculos a cabo, se debe asumir que la aceleración lineal del acelerómetro a_r es despreciable; esta aceleración puede introducir errores en el calculo de la orientación, y puede venir dada por pequeñas vibraciones o los mismos giros del sistema. Además, también debemos tomar que el acelerómetro esta colocado en paralelo a la superficie terrestre, de forma que la aceleración de la gravedad actúa únicamente sobre el eje Z. Con esto, obtendríamos el resultado de la ecuación 8.3.1.2

Teniendo lo anterior en cuenta, se plantean las matrices de rotación del alabeo (ecuación 8.3.1.2), cabeceo (ecuación 8.3.1.3) y guiñada (ecuación 8.3.1.4). Estas ecuaciones transforman un vector en una rotación del sistema de coordenadas de la figura 8.3.1.1.

$$G_p = \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = R(g - a_r) = R \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{8.3.1.2}$$

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \tag{8.3.1.3}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \tag{8.3.1.4}$$

$$R_z(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8.3.1.5}$$

Existen seis maneras posibles de ordenar estas ecuaciones, y, matemáticamente, son válidas, aunque, como se verá mas adelante, no todas sirven. Estas variables surgen al cambiar el orden en el que insertamos las ecuaciones 8.3.1.3, 8.3.1.4 y 8.3.1.5 en la ecuación 8.3.1.2. A continuación, se desarrollan las ecuaciones:

$$R_{xyz} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_x(\phi) R_y(\theta) R_z(\psi) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \tag{8.3.1.6}$$

$$\begin{pmatrix} \sin\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\theta\sin\phi \\ \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.7)$$

$$= \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \quad (8.3.1.8)$$

$$R_{yxz} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_y(\theta)R_x(\phi)R_z(\psi) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.9)$$

$$\begin{pmatrix} \cos\psi\cos\theta - \sin\theta\sin\phi\sin\psi & \sin\psi\cos\theta + \sin\theta\sin\phi\cos\psi & -\sin\theta\cos\psi \\ -\cos\phi\sin\psi & \cos\phi\cos\psi & \sin\phi \\ \cos\theta\sin\phi\sin\psi + \sin\theta\cos\psi & -\cos\psi\cos\theta\sin\phi + \sin\psi\sin\theta & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.10)$$

$$= \begin{pmatrix} -\sin\theta\cos\phi \\ \sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \quad (8.3.1.11)$$

$$R_{xzy} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_x(\phi)R_z(\psi)R_y(\theta) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.12)$$

$$\begin{pmatrix} \cos\theta\cos\psi & \sin\psi & -\cos\psi\sin\theta \\ -\cos\phi\cos\theta\sin\psi & \cos\phi\cos\psi & \cos\theta\sin\phi + \cos\phi\sin\theta\sin\psi \\ \cos\theta\sin\psi\sin\phi + \cos\psi\sin\theta & -\cos\psi\sin\phi & \cos\theta\cos\phi - \sin\theta\sin\phi\sin\psi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.13)$$

$$= \begin{pmatrix} -\cos\psi\sin\theta \\ \cos\theta\sin\phi + \cos\phi\sin\psi\sin\theta \\ \cos\phi\cos\theta - \sin\theta\sin\phi\sin\psi \end{pmatrix} \quad (8.3.1.14)$$

$$R_{yzx} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_y(\theta)R_z(\psi)R_x(\phi) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.15)$$

$$\begin{pmatrix} \cos\psi\cos\theta & \cos\phi\cos\theta\sin\psi+\sin\theta\sin\phi & \cos\theta\sin\phi\sin\psi-\sin\theta\cos\psi \\ -\sin\psi & \cos\phi\cos\psi & \cos\psi\sin\phi \\ \cos\psi\sin\theta & -\cos\theta\sin\phi+\cos\phi\sin\psi\sin\theta & \cos\theta\cos\phi+\sin\theta\sin\phi\sin\psi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.16)$$

$$= \begin{pmatrix} \cos\theta\sin\phi\sin\psi-\cos\phi\sin\theta \\ \cos\psi\sin\phi \\ \cos\theta\cos\phi+\sin\theta\sin\phi\sin\psi \end{pmatrix} \quad (8.3.1.17)$$

$$R_{zxy} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_z(\psi)R_x(\phi)R_y(\theta) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.18)$$

$$\begin{pmatrix} \cos\psi\cos\theta+\sin\theta\sin\phi\sin\psi & \cos\phi\sin\psi & \cos\theta\sin\phi\sin\psi-\sin\theta\cos\psi \\ \cos\phi\sin\theta+\cos\psi\sin\phi\sin\theta & \cos\phi\cos\psi & \cos\psi\cos\theta\sin\phi+\sin\theta\sin\psi \\ \cos\phi\sin\theta & -\sin\phi & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.19)$$

$$= \begin{pmatrix} \cos\theta\sin\phi\sin\psi-\cos\psi\sin\theta \\ \cos\psi\cos\theta\sin\phi+\sin\theta\sin\psi \\ \cos\theta\cos\phi \end{pmatrix} \quad (8.3.1.20)$$

$$R_{zyx} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = R_z(\psi)R_y(\theta)R_x(\phi) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.21)$$

$$\begin{pmatrix} \cos\psi\cos\theta & \cos\phi\sin\psi+\cos\psi\sin\phi\sin\theta & \sin\phi\sin\psi-\cos\phi\cos\psi\sin\theta \\ -\cos\theta\sin\psi & \cos\psi\cos\phi-\sin\theta\sin\phi\sin\psi & \cos\psi\sin\phi+\cos\phi\sin\psi\sin\theta \\ \sin\theta & -\cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \quad (8.3.1.22)$$

$$= \begin{pmatrix} \sin\phi\sin\psi-\cos\phi\cos\psi\sin\theta \\ \cos\psi\sin\phi+\cos\phi\sin\psi\sin\theta \\ \cos\theta\cos\phi \end{pmatrix} \quad (8.3.1.23)$$

De estos seis resultados, se pueden automáticamente descartar cuatro, los correspondientes a las ecuaciones que dependen de ψ, esto es fácilmente comprensible , ya que si el sensor rota sobre si mismo alrededor del eje Z, seguiremos obteniendo la misma lectura, de lo que deducimos que un acelerómetro no puede medir Guiñada, para ello deberemos usar el Magnetómetro, las ecuaciones para esto se detallan en la siguiente subsección.

En contraste con lo anterior, las ecuaciones 8.3.1.6 y 8.3.1.11 no dependen de ψ con lo cual, de ellas si podemos despejar ϕ y θ y obtener los ángulos de Cabeceo y Alabeo. Podemos

adoptar una cualquiera de las dos soluciones (R_{xyz} o R_{yxz}), ya que ambas son válidas e intercambiables.

Estas ecuaciones pueden suponer un problema en determinados puntos espaciales, se podría dar lo que se conoce como bloqueo cardán, una singularidad que se produce cuando el eje Y gira $\pm 90^{\circ}$, en este punto, se perdería un grado de libertad, y determinados movimientos serían indistinguibles. Pero por su construcción, el sistema objeto tiene unos movimientos limitados, y dicha situación nunca se daría.

8.3.1.1. Solución de R_{xyz}

$$\frac{G_p}{\|G_p\|} = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \Rightarrow \frac{1}{\sqrt{G_{px}^2 + G_{py}^2 + G_{pz}^2}} \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \tag{8.3.1.24}$$

$$\tan\phi_{xyz} = \frac{G_{py}}{G_{pz}} \tag{8.3.1.25}$$

$$\tan\phi_{xyz} = \frac{-G_{px}}{G_{py}\sin\phi + G_{pz}\cos\phi} = \frac{-G_{px}}{\sqrt{G_{py}^2 + G_{pz}^2}} \tag{8.3.1.26}$$

8.3.1.2. Solución de R_{yxz}

$$\frac{G_p}{\|G_p\|} = \begin{pmatrix} -\sin\theta\cos\phi \\ \sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \Rightarrow \frac{1}{\sqrt{G_{px}^2 + G_{py}^2 + G_{pz}^2}} \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} -\sin\theta\cos\phi \\ \sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \tag{8.3.1.27}$$

$$\tan\phi_{yxz} = \frac{G_{py}}{\sqrt{G_{px}^2 + G_{pz}^2}} \tag{8.3.1.28}$$

$$\tan\theta_{yxz} = \frac{-G_{px}}{G_{pz}} \tag{8.3.1.29}$$

8.3.2. Implementación de las ecuaciones

Podemos observar la implementación de las ecuaciones deducidas anteriormente en el siguiente extracto del código:

Código 8.2: Implementación de las ecuaciones que calculan los ángulos

```
Angulo_Filtrado_X = math.atan2(Accelerometer_Y,
    math.sqrt(Accelerometer_X ** 2 + Accelerometer_Z ** 2)) * Registers.Rad_Deg
Angulo_Filtrado_Y = math.atan2(-Accelerometer_X,
    math.sqrt(Accelerometer_Y ** 2 + Accelerometer_Z **2)) * Registers.Rad_Deg
```

8.3.3. Estimacion de Guiñada

Como hemos visto anteriormente, con la ayuda de un acelerómetro somos capaces de estimar con una precisión aceptable el cabeceo y el alabeo de nuestro sistema; para obtener la guiñada, debemos hacer uso del magnetómetro.

El objetivo final de este proyecto es sentar las bases para un futuro proyecto con 4 propulsores y que se sustente en el aire, así como que pueda navegar libremente a lo largo de las 3 coordenadas espaciales. La guiñada se puede definir como la desviación en grados con respecto al polo norte magnético, y, en el caso de los multicópteros, esta se corrige incrementando la velocidad de las dos hélices que giran en sentido horario y decrementando la de los antihorarios, aprovechando el torque que generan los motores por su propio movimiento.

En un sistema con únicamente dos motores, no se puede aprovechar este fenómeno físico para modificar la guiñada; no tiene sentido alguno, por lo que no se implementará.

8.3.4. Filtro Complementario

No vivimos en un mundo ideal, y, por extensión, los sensores no son ideales. En el caso del acelerómetro y giroscopio, ni se acercan a la definición de ideal.

Las medidas del acelerómetro suelen tener offsets y ruidos, como podemos ver en el gráfico 8.3.4.1, y en el giroscopio, se puede observar una deriva que se va acumulando a lo largo del tiempo (gráfico 8.3.4.2), llegando, a los pocos segundos, a mostrar medidas sin sentido alguno. Esto tiene dos posibles soluciones; el Filtro de Kalman, que estima los futuros valores de la medición, y nos da un valor limpio basado en análisis estadísticos, y el filtro complementario, mucho mas sencillo, ya que esta basado en el filtro de Kalman, pero prescinde de todo el análisis estadístico

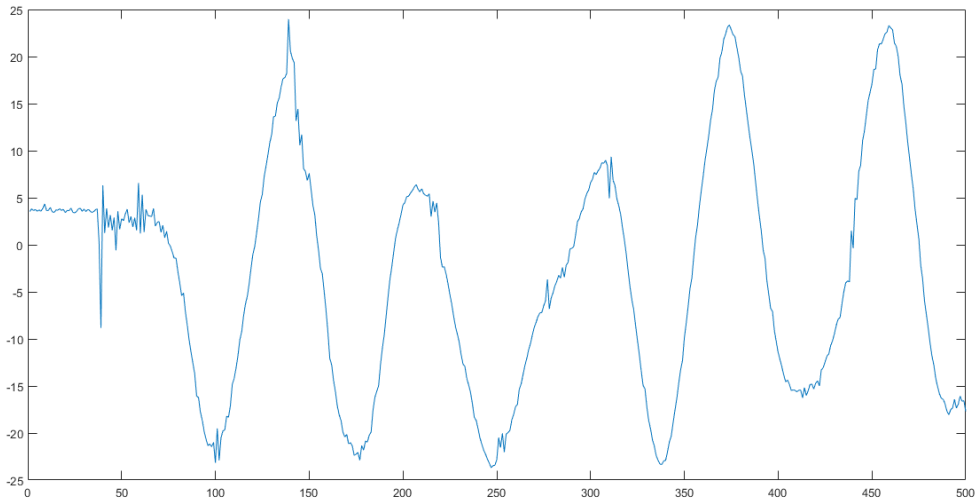


Figura 8.3.4.1 – Ruido del Acelerómetro (Eje X: Tiempo en decisegundos; Eje Y: Inclinación en grados)

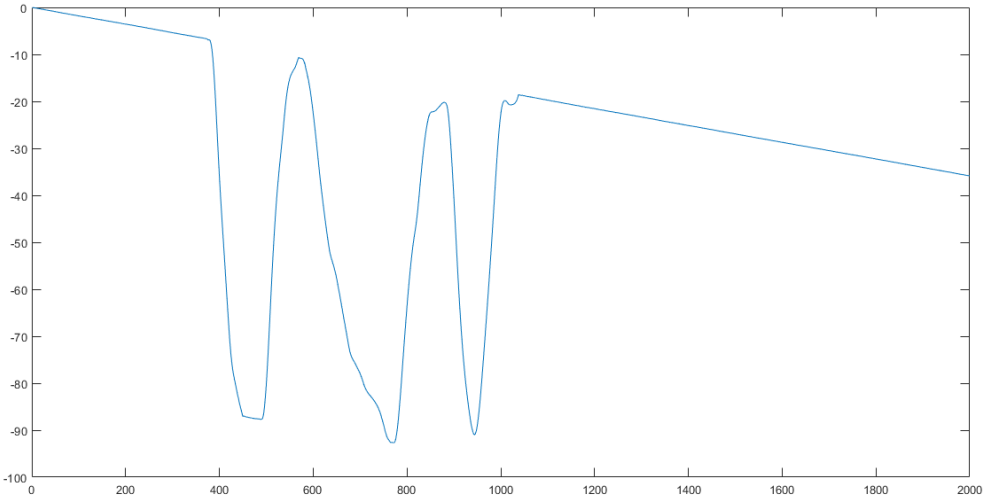


Figura 8.3.4.2 – Drift del Giroscopio (Eje X: Tiempo en decisegundos; Eje Y: Inclinación en grados)

El propósito de este filtro es eliminar tanto el ruido generado por el acelerómetro como compensar el drift del giroscopio, esto se consigue con el algoritmo que se puede ver en la ecuación 8.3.4.1.

$$Angulo = A * (Angulo + Giroscopio * \Delta t) + B * AnguloAcelerometro \tag{8.3.4.1}$$

Donde Ángulo es el angulo filtrado, A y B son los pesos del filtro, A debe acercarse a 1 y B a 0, en este proyecto, el valor de A es 0.995 y B 0.005, Giroscopio es el valor de velocidad angular devuelto por el eje correspondiente del giroscopio, y Ángulo Acelerómetro es el ángulo que forma ese eje con la horizontal, obtenido de las ecuaciones deducidas en el apartado 8.3.1

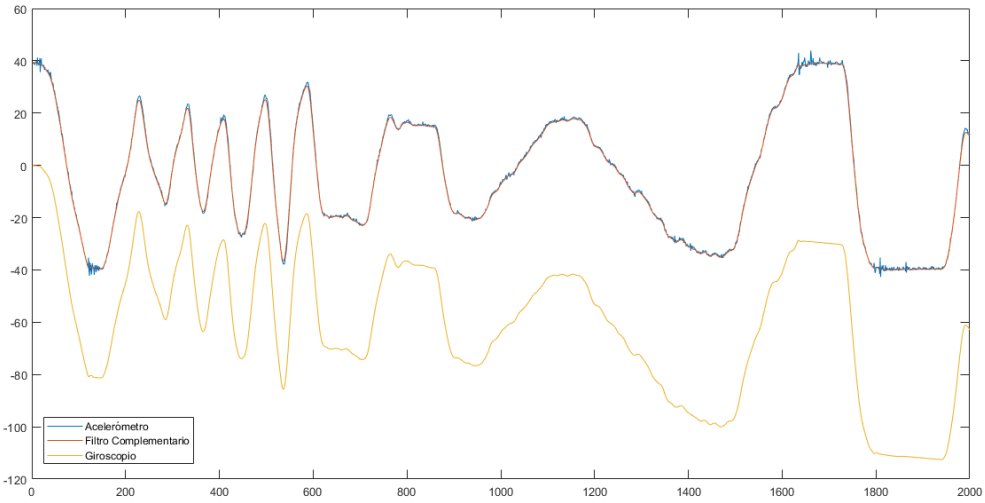


Figura 8.3.4.3 – Filtro Complementario sobre el Alabeo (Eje X: Tiempo en decisegundos; Eje Y: Inclinación en grados)

8.4. Coordinación de los PID

Es la parte menos trivial de este proyecto, ya que su origen es totalmente empírico, basado en ensayos de prueba y error. Se basa en realizar una serie de operaciones aritméticas sobre las salidas de ambos PID, como se puede observar en el diagrama de la figura 8.4.0.1 .

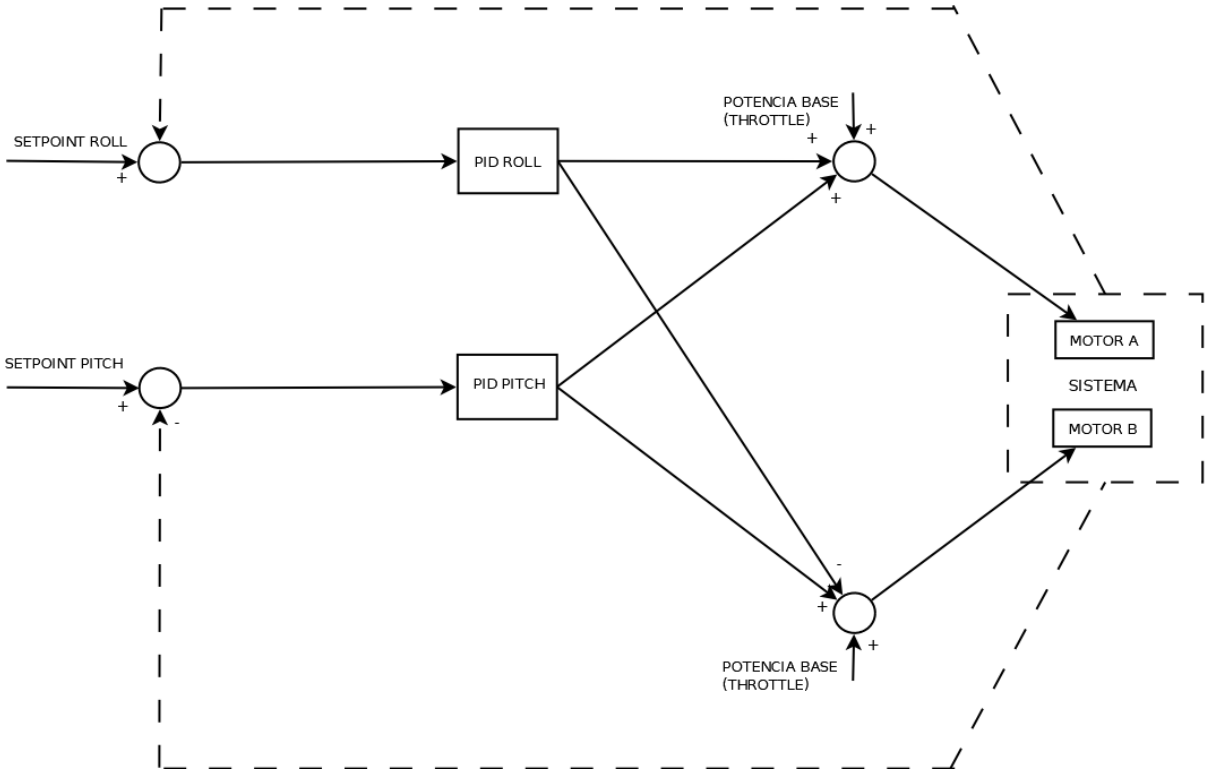


Figura 8.4.0.1 – Diagrama de Control

Código 8.3: Cálculo de la señal de salida

```
Potencia_Base        = Throttle + PID_Pitch

Ciclo_Trabajo_Negro  = Potencia_Base + PID_Roll
Ciclo_Trabajo_Dorado = Potencia_Base - PID_Roll

Ciclo_Trabajo_Negro  = (1000 + (Ciclo_Trabajo_Negro - 1000)) / 200
Ciclo_Trabajo_Dorado = (1000 + (Ciclo_Trabajo_Dorado - 1000)) / 200

file.write(str(Angulo_Filtrado_Y))
file.write('\n')

if (Ciclo_Trabajo_Negro > 7) or (Ciclo_Trabajo_Dorado > 7) :
    PWM.set_duty_cycle("P8_13", 7)
    PWM.set_duty_cycle("P8_19", 7)
    #break
else:
    PWM.set_duty_cycle("P8_13", Ciclo_Trabajo_Negro)
```

En la primera línea del código, se suma la salida del PID que controla el Cabeceo a Throttle, Throttle es la potencia que un piloto le daría al drone para ascender o descender, es decir, varía entre en 0 y el 100 % de la PPM (La librería de Python que escribe la PPM escribe realmente una PWM de 50Hz, pero cuyo valor mínimo es de 1ms y máximo de 2ms). En este proyecto, será el 20 %, ya que es aproximadamente el valor de potencia que hace equilibrar el sistema en el Cabeceo teniendo en cuenta el contrapeso. Esta suma permite ajustar finamente la potencia base, de forma que se mantenga estable.

La segunda y tercera línea, simplemente suman y restan los valores del PID que controla el Alabeo a lo calculado anteriormente.

A continuación, se efectúan una serie de operaciones con los valores obtenidos, de forma que se transforman en un número que oscila entre 5 y 10 y que sirve para escribir en la PPM.

La función If limita la potencia asignada a los motores, si sobrepasa cierto valor, se satura la salida, ya que altas velocidades pueden resultar peligrosas en un sistema fijo como este, esta medida de seguridad, al igual que los protectores de hélices, fue tomada en previsión de un posible accidente.

8.5. El algoritmo de control PID

En una sección anterior, se ha hablado sobre y descrito el algoritmo de control PID, por lo que esta sección está dedicada exclusivamente a la implementación del mismo en un entorno Python y al ajuste de las constantes.

El presente proyecto consta de dos algoritmos de control PID que trabajan conjuntamente con el objetivo de estabilizarlo en los dos ángulos de libertad sobre los que al final hemos trabajado. En primer lugar, se ha implementado un único PID, bloqueando el movimiento de cabeceo, y una vez ajustados los parámetros de este, se ha implementado el segundo.

8.5.1. Implementación del PID del alabeo

Podemos ver, en las siguientes líneas, el código que se corresponde únicamente al algoritmo de control PID del alabeo. Consiste en una discretización estándar del algoritmo de control PID, pero que ha dado buenos resultados en la práctica. Se calcula en primer lugar el error, y posteriormente, haciendo uso de las constantes, cada uno de los términos; Proporcional, Integral y Derivativo, para al final sumarlos y obtener la salida.

Código 8.4: PID del Alabeo Implementado

```
Error_Pitch          = Angulo_Filtrado_Y - Angulo_Estabilidad_Pitch

Proporcional_Pitch = Kp_P * Error_Pitch
Integral_Pitch     = Integral_Pitch + (Ki_P * (Error_Pitch + Error_Anterior_Pitch))
Derivativo_Pitch   = Kd_P * (Error_Pitch - Error_Anterior_Pitch)
PID_Pitch          = Proporcional_Pitch + Integral_Pitch + Derivativo_Pitch

Error_Anterior_Pitch = Error_Pitch
```

Como se puede ver, es una implementación sencilla, pero la complejidad se sitúa en el ajuste de las constantes. No se puede utilizar ningún método empírico de sintonización de PID estudiados en Ingeniería de Control, ya que el sistema no es lineal, tiene unos margenes de movimiento limitados y se inestabiliza con mucha facilidad. Esto nos lleva a la necesidad de ajustarlo manualmente, observando la respuesta del sistema ante las diferentes modificaciones y actuando sobre las constantes en función de esta.

Para llevar a cabo el ajuste manual, me guiaré por la tabla 8.5.1.1 y por las siguientes reglas heurísticas de ajuste:

1. Acción Proporcional
    - Tiempo integral (Ti), a su máximo valor.
    - Tiempo derivativo (Td), a su mínimo valor.
    - Incrementar la ganancia (Kp) hasta obtener las características de respuesta deseadas

2. Acción Integral
    - Reducir Ti hasta anular el error en estado estacionario, aunque la oscilación sea excesiva.
    - Disminuir ligeramente la ganancia.
    - Repetir hasta obtener las características de respuesta deseadas.
3. Acción Derivativa
    - Mantener ganancia y tiempo integral obtenidos anteriormente.
    - Aumentar Td hasta obtener características similares pero con la respuesta más rápida
    - Aumentar ligeramente la ganancia si fuera necesario.

	Kp Aumenta	**Ti Disminuye**	**Td Aumenta**
Estabilidad	Disminuye	Disminuye	Aumenta
Velocidad	Aumenta	Aumenta	Aumenta
Error Estacionario	No Eliminado	Eliminado	No Eliminado
Perturbación Control	Aumenta Rápido	Aumenta Despacio	Aumenta Rápido

Tabla 8.5.1.1 – Como afecta la modificación de las constantes del PID a la respuesta del sistema

Siguiendo estas reglas de ajuste se ha conseguido estabilizar el eje correspondiente al Alabeo con una respuesta realmente buena, como podemos observar en la imagen .

8.5.2. Implementación del PID del cabeceo

El PID del cabeceo es exactamente igual al del alabeo en lo que a implementación se refiere, únicamente cambia la forma en la que actúa sobre la salida final, como se ha especificado en la sección correspondiente.

El ajuste se ha llevado a cabo siguiendo las mismas reglas que el apartado anterior, obteniendo, una vez alcanzadas las constantes adecuadas, la respuesta ideal.

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

ANEXOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice del documento ANEXOS

9 Documentación de partida

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Bello Corbeira,Constantino*
APELIDOS E NOME:

DNI: [redacted] **Fecha de Solicitud:** Feb2017
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Estabilización de un sistema aeropropulsado

Número TFG: *770G01A120*

TUTOR:(Titor) *Calvo Rolle,Jose Luis*

COTUTOR/CODIRECTOR: *Óscar Fontenla Romero*

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento*:*

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Jueves, 30 de Marzo del 2017

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: Bello Corbeira,Constantino

10 Códigos de programación

10.1. Filtro Complementario

Código 10.1: Filtro Complementario

```
# Importamos todas las liberias necesarias para ejecutar el codigo:
from   Adafruit_I2C      import Adafruit_I2C
from   time              import time
from   time              import sleep
import Registers
import math

# Configuracion inicial del bus I2C que direcciona los registros
# (Asignacion de  la direccion I2C):
I2C_MPU9250 = Adafruit_I2C(0x68,2)

# Estas dos lineas inferiores configuran los registros del MPU9250 de forma que
# podemos acceder a los registros del AK8963, la segunda linea permite que los
#pines ES_CL y ES_DA entren en modo bypass:

I2C_MPU9250.write8(Registers.PWR_MGMT_1,  0)
I2C_MPU9250.write8(Registers.INT_PIN_CFG, 2)

# Configuracion inicial del bus I2C que direcciona los registros
# (Asignacion de la direccion I2C):

I2C_AK8963  = Adafruit_I2C(0x0C,2)

I2C_AK8963.write8(Registers.CNTL1, 1)

# Esta funcion lee el bus I2C, y nos devuelve el valor leido entre el valor a
# fondo de escala positivo y negativo:

def Read_I2C (MSB_Register, LSB_Register, Full_Scale_Value, MPU9250_AK8963) :

    if MPU9250_AK8963 == 0 :
        High_Byte = I2C_MPU9250.readU8(MSB_Register)
```

```
        Low_Byte  = I2C_MPU9250.readU8(LSB_Register)
    else :
        High_Byte = I2C_AK8963.readU8(MSB_Register)
        Low_Byte  = I2C_AK8963.readU8(LSB_Register)

    High_Byte = High_Byte << 8
    Read_Value = High_Byte | Low_Byte
    Read_Value = (Read_Value/32768.0) * Full_Scale_Value

    if Read_Value >= Full_Scale_Value  :
        Read_Value = Read_Value - (2 * Full_Scale_Value)

    return Read_Value

# Defino valores iniciales:

Tiempo = 0
i      = 0
Angulo_Giroscopio = 0

# Abro los ficheros que voy a escribir, en modo de lectura:

file = open("Filtro_Complementario.txt","w")
file_drift = open("Drift_Giroscopio.txt","w")

while i < 400:
    Tiempo_Inicial = time()

    i = i + 1

    # Llamadas a funcion que leen valores concretos en el bus I2C:

    Accelerometer_X = Read_I2C (Registers.ACCEL_XOUT_H, Registers.ACCEL_XOUT_L,
                      Registers.Accelerometer_Full_Scale_Value, 0)
    Accelerometer_Y = Read_I2C (Registers.ACCEL_YOUT_H, Registers.ACCEL_YOUT_L,
                      Registers.Accelerometer_Full_Scale_Value, 0)
    Accelerometer_Z = Read_I2C (Registers.ACCEL_ZOUT_H, Registers.ACCEL_ZOUT_L,
                      Registers.Accelerometer_Full_Scale_Value, 0)

    Gyroscope_X =     Read_I2C (Registers.GYRO_XOUT_H, Registers.GYRO_XOUT_L,
                      Registers.Gyroscope_Full_Scale_Value, 0)
    Gyroscope_Y =     Read_I2C (Registers.GYRO_YOUT_H, Registers.GYRO_YOUT_L,
                      Registers.Gyroscope_Full_Scale_Value, 0)
    Gyroscope_Z =     Read_I2C (Registers.GYRO_ZOUT_H, Registers.GYRO_ZOUT_L,
                      Registers.Gyroscope_Full_Scale_Value, 0)



    Angulo_Giroscopio = Angulo_Giroscopio + Gyroscope_X * 0.02
```

```
    # Filtro complementario:

    Angulo_Alabeo = math.atan2(Accelerometer_Y,
                    math.sqrt(Accelerometer_X ** 2 + Accelerometer_Z ** 2)) *
                    Registers.Rad_Deg

    # Soluciona los problemas que da la primera iteracion del bucle, al no
    # existir valores anteriores:

    if i == 1 :
        Angulo_Filtrado_Alabeo = Angulo_Alabeo
    else :
        Angulo_Filtrado_Alabeo = 0.995 * (Angulo_Filtrado_Alabeo +
                                 Gyroscope_X * 0.02) + 0.005 * Angulo_Alabeo

    # Escrituras en fichero

    file.write(str(Angulo_Alabeo))
    file.write(' ')
    file.write(str(Angulo_Filtrado_Alabeo))
    file.write(' ')
    file.write(str(Angulo_Giroscopio))
    file.write('\n')

    file_drift.write(str(Angulo_Giroscopio))
    file_drift.write('\n')

    # Asegura un tiempo de muestreo constante

    Tiempo_Ejecucion = time() - Tiempo_Inicial
    sleep (0.02 - Tiempo_Ejecucion)

#Cerramos los ficheros:

file.close()
file_drift.close()
```

10.2. Registros

Código 10.2: Registros

```
# Definiciones Generales

PI                            = 3.14159265359
Rad_Deg                       = 57.29577951
Accelerometer_Full_Scale_Value = 2 # G Puede ser 2, 4, 8 o 16 (MPU9250)
```

```
Gyroscope_Full_Scale_Value     = 250 # DPS (Puede ser 250, 500, 1000 o 2000)
Magnetometer_Full_Scale_Value  = 4800 # Microtesla

# Definicion de registros de MPU9250

SELF_TEST_X_GYRO   = 0x00
SELF_TEST_Y_GYRO   = 0x01
SELF_TEST_Z_GYRO   = 0x02
SELF_TEST_X_ACCEL  = 0x0D
SELF_TEST_Y_ACCEL  = 0x0E
SELF_TEST_Z_ACCEL  = 0x0F
XG_OFFSET_H        = 0x13
XG_OFFSET_L        = 0x14
YG_OFFSET_H        = 0x15
YG_OFFSET_L        = 0x16
ZG_OFFSET_H        = 0x17
ZG_OFFSET_L        = 0x18
SMPLRT_DIV         = 0x19
CONFIG             = 0x1A
GYRO_CONFIG        = 0x1B
ACCEL_CONFIG       = 0x1C
CONFIG_2           = 0x1D
LP_ACCEL_ODR       = 0x1E
WOM_THR            = 0x1F
FIFO_EN            = 0x23
I2C_MST_CTRL       = 0x24
I2C_SLV0_ADDR      = 0x25
I2C_SLV0_REG       = 0x26
I2C_SLV0_CTRL      = 0x27
I2C_SLV1_ADDR      = 0x28
I2C_SLV1_REG       = 0x29
I2C_SLV1_CTRL      = 0x2A
I2C_SLV2_ADDR      = 0x2B
I2C_SLV2_REG       = 0x2C
I2C_SLV2_CTRL      = 0x2D
I2C_SLV3_ADDR      = 0x2E
I2C_SLV3_REG       = 0x2F
I2C_SLV3_CTRL      = 0x30
I2C_SLV4_ADDR      = 0x31
I2C_SLV4_REG       = 0x32
I2C_SLV4_DO        = 0x33
I2C_SLV4_CTRL      = 0x34
I2C_SLV4_DI        = 0x35
I2C_MST_STATUS     = 0x36
INT_PIN_CFG        = 0x37
INT_ENABLE         = 0x38
INT_STATUS         = 0x3A
ACCEL_XOUT_H       = 0x3B
ACCEL_XOUT_L       = 0x3C
ACCEL_YOUT_H       = 0x3D
```

```
ACCEL_YOUT_L       = 0x3E
ACCEL_ZOUT_H       = 0x3F
ACCEL_ZOUT_L       = 0x40
TEMP_OUT_H         = 0x41
TEMP_OUT_L         = 0x42
GYRO_XOUT_H        = 0x43
GYRO_XOUT_L        = 0x44
GYRO_YOUT_H        = 0x45
GYRO_YOUT_L        = 0x46
GYRO_ZOUT_H        = 0x47
GYRO_ZOUT_L        = 0x48
EXT_SENS_DATA_00   = 0x49
EXT_SENS_DATA_01   = 0x4A
EXT_SENS_DATA_02   = 0x4B
EXT_SENS_DATA_03   = 0x4C
EXT_SENS_DATA_04   = 0x4D
EXT_SENS_DATA_05   = 0x4E
EXT_SENS_DATA_06   = 0x4F
EXT_SENS_DATA_07   = 0x50
EXT_SENS_DATA_08   = 0x51
EXT_SENS_DATA_09   = 0x52
EXT_SENS_DATA_10   = 0x53
EXT_SENS_DATA_11   = 0x54
EXT_SENS_DATA_12   = 0x55
EXT_SENS_DATA_13   = 0x56
EXT_SENS_DATA_14   = 0x57
EXT_SENS_DATA_15   = 0x58
EXT_SENS_DATA_16   = 0x59
EXT_SENS_DATA_17   = 0x5A
EXT_SENS_DATA_18   = 0x5B
EXT_SENS_DATA_19   = 0x5C
EXT_SENS_DATA_20   = 0x5D
EXT_SENS_DATA_21   = 0x5E
EXT_SENS_DATA_22   = 0x5F
EXT_SENS_DATA_23   = 0x60
I2C_SLV0_DO        = 0x63
I2C_SLV1_DO        = 0x64
I2C_SLV2_DO        = 0x65
I2C_SLV3_DO        = 0x66
I2C_MST_DELAY_CTRL = 0x67
SIGNAL_PATH_RESET  = 0x68
MOT_DETECT_CTRL    = 0x69
USER_CTRL          = 0x6A
PWR_MGMT_1         = 0x6B
PWR_MGMT_2         = 0x6C
FIFO_COUNTH        = 0x72
FIFO_COUNTL        = 0x73
FIFO_R_W           = 0x74
WHO_AM_I           = 0x75
XA_OFFSET_H        = 0x77
```

```
XA_OFFSET_L         = 0x78
YA_OFFSET_H         = 0x7A
YA_OFFSET_L         = 0x7B
ZA_OFFSET_H         = 0x7D
ZA_OFFSET_L         = 0x7E

# Definicion de registros de AK8963

WIA                 = 0x00
INFO                = 0x01
ST1                 = 0x02
HXL                 = 0x03
HXH                 = 0x04
HYL                 = 0x05
HYH                 = 0x06
HZL                 = 0x07
HZH                 = 0x08
ST2                 = 0x09
CNTL1               = 0x0A
CNTL2               = 0x0B
ASTC                = 0x0C
TS1                 = 0x0D
TS2                 = 0x0E
I2CDIS              = 0x0F
ASAX                = 0x10
ASAY                = 0x11
ASAZ                = 0x12
```

10.3. Inicialización

Código 10.3: Inicialización

```
# Importamos la libreria necesaria para controlar las salidas PWM
import Adafruit_BBIO.PWM as PWM
# Y las librerias para permitir controlar los tiempos.
from time import sleep
from time import time
def Initialization_ESC () :
    # La secuencia de inicializacion es como sigue; en primer lugar, se escribe
    # una PWM de un ciclo de trabajo del 100 %, a continuacion, se conecta la
    # fuente de alimentacion, los ESC emitiran 3 pitidos que indican que se ha
    # conectado alimentacion satisfactoriamente. Una vez hecho esto, se escribe
    # un ciclo de trabajo del 0 %, que debido a la configuracion de los ESC, este
    # ciclo se corresponde a una PWM cuyo ciclo de trabajo es del 5.74 %
    # Escribo una PWM de 9.2 % de ciclo de trabajo, ya que esto se corresponde al
```

```
    PWM.start("P8_13", 10, 50) # Habilita el pin 13 (PWM1A)
    PWM.start("P8_19", 10, 50) # Habilita el pin 19 (PWM1B)

    print ("Conecte la alimentacion y presione enter, en ese orden")
    raw_input()

    sleep (1)

    # En el caso que sigue, utilizo 5.7 en lugar de 5.74 debido a que se
    # observaron ligeros movimientos de los motores en este ultimo valor, esto
    # asegura que el valor que interpretan los ESC es 0.

    PWM.set_duty_cycle("P8_13", 5)
    PWM.set_duty_cycle("P8_19", 5)

    sleep (5)

#Initialization_ESC ()
#while True :
#    ciclo = input()
#    PWM.set_duty_cycle("P8_13", ciclo)
#    PWM.set_duty_cycle("P8_19", ciclo)
```

10.4. PID Roll y PID Pitch

Código 10.4: Roll y Pitch

```
# Importamos las librerias de Adafruit que nos posibilitan el control del bus
# I2C y de las salidas PWM.
from   Adafruit_I2C      import Adafruit_I2C
from   time              import time
import Adafruit_BBIO.PWM as     PWM
import math
import Registers
import Initialization
# Utilizamos un wildcard para importar todas las variables de Registers y de
# math, de forma que las podremos usar con mayor simplicidad. El uso seria
# Registers.Variable o math.funcion
from Registers      import*
from math           import*
from Initialization import*

# Configuracion inicial del bus I2C que direcciona los registros (Asignacion de
# la direccion I2C)
I2C_MPU9250 = Adafruit_I2C(0x68,2)
```

```
# Estas dos lineas inferiores configuran los registros del MPU9250 de forma que
# podemos acceder a los registros del AK8963, la segunda linea permite que los
#pines ES_CL y ES_DA entren en modo bypass,
I2C_MPU9250.write8(Registers.PWR_MGMT_1,  0)
I2C_MPU9250.write8(Registers.INT_PIN_CFG, 2)

# Configuracion inicial del bus I2C que direcciona los registros (Asignacion de
# la direccion I2C), pero para el magnetometro, ya que debemos haber habilitado
# el modo bypass previamente.

I2C_AK8963  = Adafruit_I2C(0x0C,2)
I2C_AK8963.write8(Registers.CNTL1, 1)

Initialization.Initialization_ESC ()

# Funciones

# Funcion que lee un registro
#-------------------------------------------------------------------------------
def Read_I2C (MSB_Register, LSB_Register, Full_Scale_Value, MPU9250_AK8963) :
# Esta funcion tiene varias entradas, las dos primeras le indican que registro
# debe leer, la tercera, el valor a fondo de escala, que se utilizara para saber
# la escala en la que la IMU nos devuelve el valor, y la cuarta, que es un "bit"
# que especifica si se va a leer el acelerometro/giroscopio o el magnetometro.
    if MPU9250_AK8963 == 0 :
        High_Byte = I2C_MPU9250.readU8(MSB_Register)
        Low_Byte  = I2C_MPU9250.readU8(LSB_Register)
    else :
        High_Byte = I2C_AK8963.readU8(MSB_Register)
        Low_Byte  = I2C_AK8963.readU8(LSB_Register)
# Las siguientes lineas desplazan y encadenan el byte mas significativo y el
# menos significativo, de forma que nos queda un resultado de 16 bits, que en la
# tercera linea convertimos a un valor en unidades de aceleracion, velocidad
# angular o intensidad de campo magnetico
    High_Byte = High_Byte << 8
    Read_Value = High_Byte | Low_Byte
    Read_Value = (Read_Value/32768.0) * Full_Scale_Value
# Estas lineas de codigo aseguran que el valor devuelto sea positivo, y si no
# lo es, el signo se modifica, ya que el bit mas significativo es un bit de
# signo
    if Read_Value >= Full_Scale_Value  :
        Read_Value = Read_Value - (2 * Full_Scale_Value)

    return Read_Value
#-------------------------------------------------------------------------------

# Declaracion de variables que van a ser utilizadas en el programa por los PID
#-------------------------------------------------------------------------------
# Variables especiales
#-------------------------------
```

```
Throttle                  = 1210
Angulo_Estabilidad_Roll   = 0
Angulo_Estabilidad_Pitch = 0
#-------------------------------------

# Variables del PID
#-------------------------------------
Kp_R = 1.41
Ti_R = 19
Td_R = 0.40
Kp_P = 1.70
Ti_P = 40.0
Td_P = 0.35
Ts   = 0.01

Ki_R = (Kp_R * Ts) / Ti_R
Kd_R = (Kp_R * Td_R) / Ts

Ki_P = (Kp_P * Ts) / Ti_P
Kd_P = (Kp_P * Td_P) / Ts
#-------------------------------------
#-------------------------------------------------------------------------------

# Otras variables necesarias para la correcta ejecucion del programa en la
# primera iteracion del bucle
#-------------------------------------------------------------------------------
# Esta linea impide que en la primera iteracion del bucle while salgan valores
# extranos
Tiempo_Anterior      = time()
Tiempo_Ejecucion     = 0.01
Error_Anterior_Roll  = 0.0
Error_Anterior_Pitch = 0.0
Integral_Roll        = 0.0
Integral_Pitch       = 0.0
#-------------------------------------------------------------------------------

# Calculo del angulo inicial, al estar en una posicion estatica, podemos tomar
# simplemente el valor dado por el acelerometro
#-------------------------------------------------------------------------------
Accelerometer_X = Read_I2C (Registers.ACCEL_XOUT_H, Registers.ACCEL_XOUT_L,
    Registers.Accelerometer_Full_Scale_Value, 0)
Accelerometer_Y = Read_I2C (Registers.ACCEL_YOUT_H, Registers.ACCEL_YOUT_L,
    Registers.Accelerometer_Full_Scale_Value, 0)
Accelerometer_Z = Read_I2C (Registers.ACCEL_ZOUT_H, Registers.ACCEL_ZOUT_L,
    Registers.Accelerometer_Full_Scale_Value, 0)

Angulo_Filtrado_X = math.atan2(Accelerometer_Y,
    math.sqrt(Accelerometer_X ** 2 + Accelerometer_Z ** 2)) * Registers.Rad_Deg
Angulo_Filtrado_Y = math.atan2(-Accelerometer_X,
    math.sqrt(Accelerometer_Y ** 2 + Accelerometer_Z **2)) * Registers.Rad_Deg
```

```
#-------------------------------------------------------------------------------
# Abrimos un fichero en el que volcaremos datos para posteriormente
# visualizarlos en un grafico
file = open("PID.txt","w")
# Bucle que se ejecuta continuamente
while True :

# Las linea siguiente y las finales se utilizan para calcular el tiempo de una
# iteracion del bucle while
    Tiempo = time()

    Accelerometer_X = Read_I2C (Registers.ACCEL_XOUT_H, Registers.ACCEL_XOUT_L,
        Registers.Accelerometer_Full_Scale_Value, 0)
    Accelerometer_Y = Read_I2C (Registers.ACCEL_YOUT_H, Registers.ACCEL_YOUT_L,
        Registers.Accelerometer_Full_Scale_Value, 0)
    Accelerometer_Z = Read_I2C (Registers.ACCEL_ZOUT_H, Registers.ACCEL_ZOUT_L,
        Registers.Accelerometer_Full_Scale_Value, 0)

    Gyroscope_X =     Read_I2C (Registers.GYRO_XOUT_H, Registers.GYRO_XOUT_L,
        Registers.Gyroscope_Full_Scale_Value, 0)
    Gyroscope_Y =     Read_I2C (Registers.GYRO_YOUT_H, Registers.GYRO_YOUT_L,
        Registers.Gyroscope_Full_Scale_Value, 0)
    #Gyroscope_Z =     Read_I2C (Registers.GYRO_ZOUT_H, Registers.GYRO_ZOUT_L,
    #    Registers.Gyroscope_Full_Scale_Value, 0)

    #Magnetometer_X =  Read_I2C (Registers.HXH, Registers.HXL,
    #    Registers.Magnetometer_Full_Scale_Value, 1)
    #Magnetometer_Y =  Read_I2C (Registers.HYH, Registers.HYL,
    #    Registers.Magnetometer_Full_Scale_Value, 1)
    #Magnetometer_Z =  Read_I2C (Registers.HZH, Registers.HZL,
    #    Registers.Magnetometer_Full_Scale_Value, 1)

    Angulo_Aceleracion_X = math.atan2(Accelerometer_Y,
        math.sqrt(Accelerometer_X ** 2 + Accelerometer_Z ** 2)) * Registers.Rad_Deg
    Angulo_Filtrado_X    = 0.995 * (Angulo_Filtrado_X + Gyroscope_X * 0.01) +
    0.005 * Angulo_Aceleracion_X

    Angulo_Aceleracion_Y = math.atan2(-Accelerometer_X,
        math.sqrt(Accelerometer_Y ** 2 + Accelerometer_Z **2)) * Registers.Rad_Deg
    Angulo_Filtrado_Y    = 0.995 * (Angulo_Filtrado_Y + Gyroscope_Y * 0.01) +
    0.005 * Angulo_Aceleracion_Y

    # Si queremos actuar unicamente sobre el Roll debemos comentar la seccion
    # PID PITCH y eliminar o comentar PID_Pitch en la linea 188 del codigo.

    #PID PITCH
    #-------------------------------------------------------------------------------
    Error_Pitch          = Angulo_Filtrado_Y - Angulo_Estabilidad_Pitch

    Proporcional_Pitch = Kp_P * Error_Pitch
```

```
    Integral_Pitch     = Integral_Pitch + (Ki_P * (Error_Pitch + Error_Anterior_Pitch))
    Derivativo_Pitch   = Kd_P * (Error_Pitch - Error_Anterior_Pitch)
    PID_Pitch          = Proporcional_Pitch + Integral_Pitch + Derivativo_Pitch

    Error_Anterior_Pitch = Error_Pitch
    #-------------------------------------------------------------------------
    #PID ROLL
    #-------------------------------------------------------------------------
    Error_Roll          = Angulo_Filtrado_X - Angulo_Estabilidad_Roll

    Proporcional_Roll = Kp_R * Error_Roll
    Integral_Roll     = Integral_Roll + (Ki_R * (Error_Roll + Error_Anterior_Roll))
    Derivativo_Roll   = Kd_R * (Error_Roll - Error_Anterior_Roll)
    PID_Roll          = Proporcional_Roll  + Integral_Roll + Derivativo_Roll

    Error_Anterior_Roll = Error_Roll
    #-------------------------------------------------------------------------

    Potencia_Base        = Throttle + PID_Pitch

    Ciclo_Trabajo_Negro  = Potencia_Base + PID_Roll
    Ciclo_Trabajo_Dorado = Potencia_Base - PID_Roll

    Ciclo_Trabajo_Negro  = (1000 + (Ciclo_Trabajo_Negro - 1000)) / 200
    Ciclo_Trabajo_Dorado = (1000 + (Ciclo_Trabajo_Dorado - 1000)) / 200

    file.write(str(Angulo_Filtrado_Y))
    file.write('\n')

    if (Ciclo_Trabajo_Negro > 7) or (Ciclo_Trabajo_Dorado > 7) :
        PWM.set_duty_cycle("P8_13", 7)
        PWM.set_duty_cycle("P8_19", 7)
        #break
    else:
        PWM.set_duty_cycle("P8_13", Ciclo_Trabajo_Negro)
        PWM.set_duty_cycle("P8_19", Ciclo_Trabajo_Dorado)


    Tiempo_Ejecucion = Tiempo - Tiempo_Anterior
    if Tiempo_Ejecucion < 0.01 :
        sleep (0.01 - Tiempo_Ejecucion)

    Tiempo_Anterior = Tiempo

file.close()
```

11 Otros anexos

11.1. Configuración inicial de la Beaglebone Black

La BBB se ha recibido con un sistema operativo instalado, pero dicho sistema no está actualizado, además de ser una versión antigua de Debian. Por ello, se pretenderá actualizar el software de forma que se pueda contar con las últimas actualizaciones en lo que a seguridad y estabilidad se refiere. El equipo de desarrolladores y colaboradores del proyecto ha puesto a disposición una página web [I] de la que se pueden descargar las últimas versiones de los sistemas operativos disponibles para la placa.

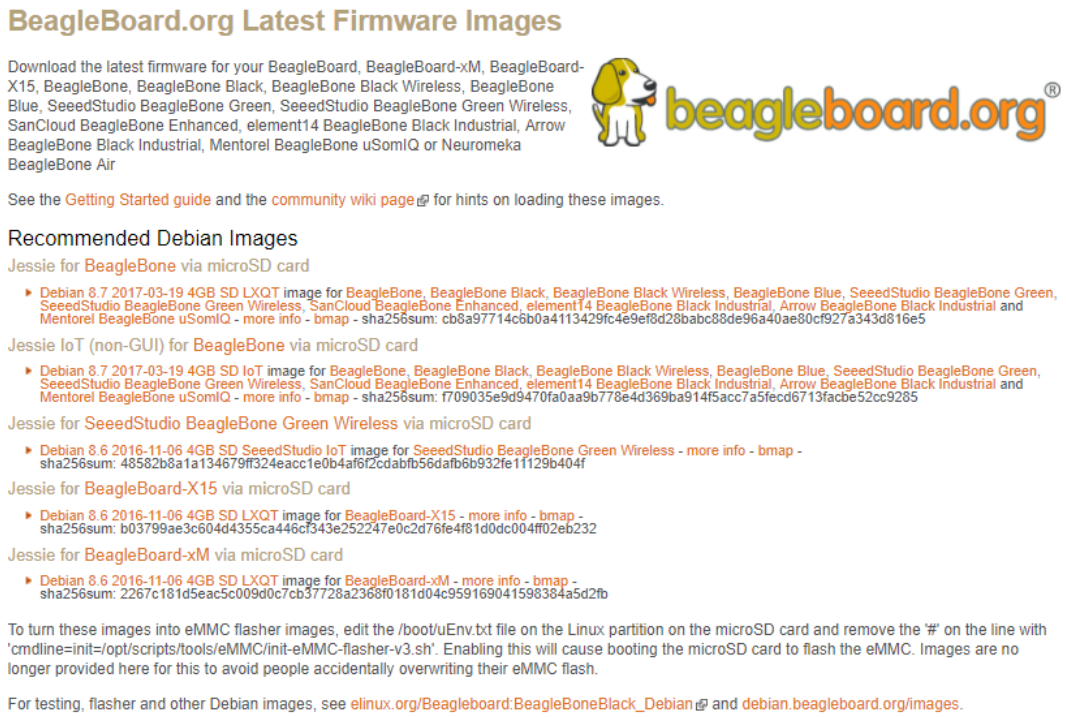


Figura 11.1.0.1 – Vista de la web de descarga de imágenes a 11/07/2017

De esta página web, obtenemos la imagen deseada, en nuestro caso, será Debian 8.7 2017-03-19 4GB SD LXQT. Como resultado de esta descarga, obtenemos un archivo comprimido, cuyo peso ronda los 700MB, que debemos descomprimir en un directorio de nuestro ordenador y grabar en una tarjeta micro SD de al menos 4GB. Dicho proceso no es trivial, y se necesita echar mano de herramientas ajenas al sistema operativo, en nuestro caso, 7zip [II] y Win32Imager [III].

7zip es un software libre de extracción y compresión de ficheros, lanzado en el año 1999 por Ígor Pávlov. La última versión del software puesta a disposición del público fué la 16.04, en el año 2016. El proyecto sigue en funcionamiento.

Win32Imager es una herramienta para sistemas Windows con la licencia *GNU General Public License Version 2.0 (GPLv2)*, es un software sencillo cuyo principal uso es la escritura de imágenes de disco en formato ISO en dispositivos portátiles, como memorias USB o, en nuestro caso, una tarjeta Micro SD. Esto permite la lectura de dichas imágenes por dispositivos que carecen de una unidad de CD, como es el caso de la Beaglebone.

Se procederá a la descarga del firmware Debian y de ambos programas, instalando los dos últimos en el sistema Windows. Con 7zip se extraerá la imagen ISO y se grabará en la tarjeta micro SD con Win32Imager.

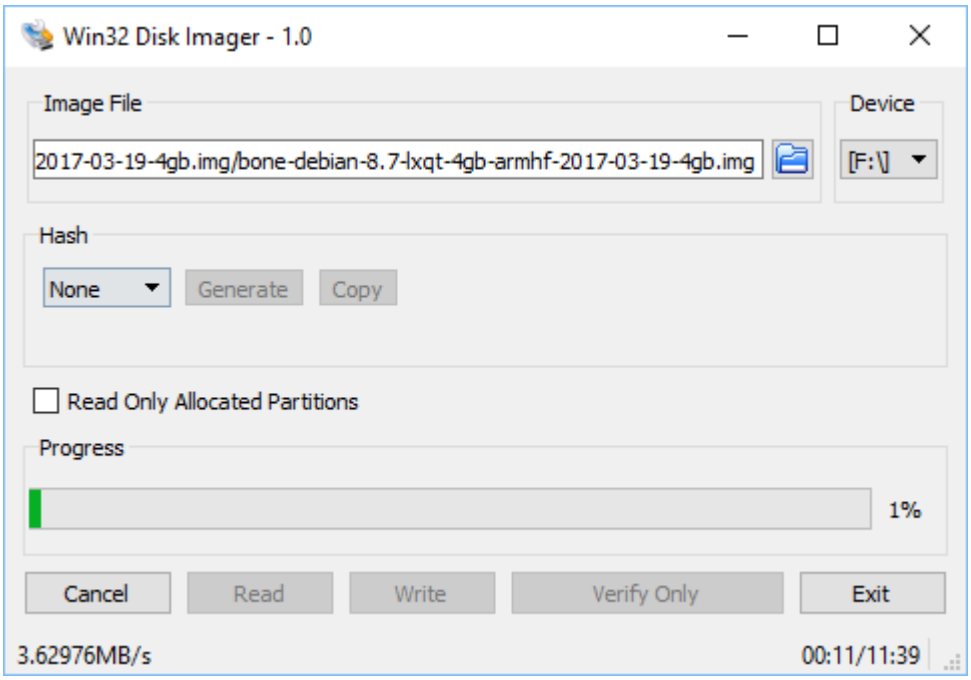


Figura 11.1.0.2 – Escritura de la imagen .iso en la tarjeta Micro SD

La imagen descargada no permite, en principio, ser flasheada en la memoria interna de la BBB, pero existen dos formas para llevar esto a cabo:

- Editar el archivo *uEnv.txt* que se encuentra localizado en la carpeta */boot* de la tarjeta, de esta forma, al iniciar la BBB desde la tarjeta Micro SD, el sistema operativo se instalará de forma automática.

- Iniciar el sistema operativo desde la Micro SD y ejecutar en una consola de comandos, con privilegios de administrador, las siguientes órdenes:

  1. *cd /opt/scripts/tools/eMMC/*
  2. *sudo ./init-eMMC-flasher-v3.sh*

  La primera orden nos sitúa en la carpeta en la que se encuentra alojado el *flasher*, un

script del sistema operativo encargado de instalar en la memoria interna de la BBB Debian, la segunda orden ejecuta dicho script.

Después de haber buscado, sin éxito, el archivo del que se habla en la primera opción, me he decantado por la segunda opción, que a pesar de ser mas laboriosa, funciona correctamente.

Para llevar esto a cabo, se puede conectar un teclado y ratón a la BBB, además de una pantalla, pero esta versión de Debian diseñada exclusivamente para BBB cuenta con un entorno de desarrollo integrado llamado Cloud9, que, por supuesto, permite ejecutar órdenes en la consola de Debian, a dicho entorno se puede acceder introduciendo la siguiente dirección en nuestro navegador Web: *192.168.7.2:3000*.

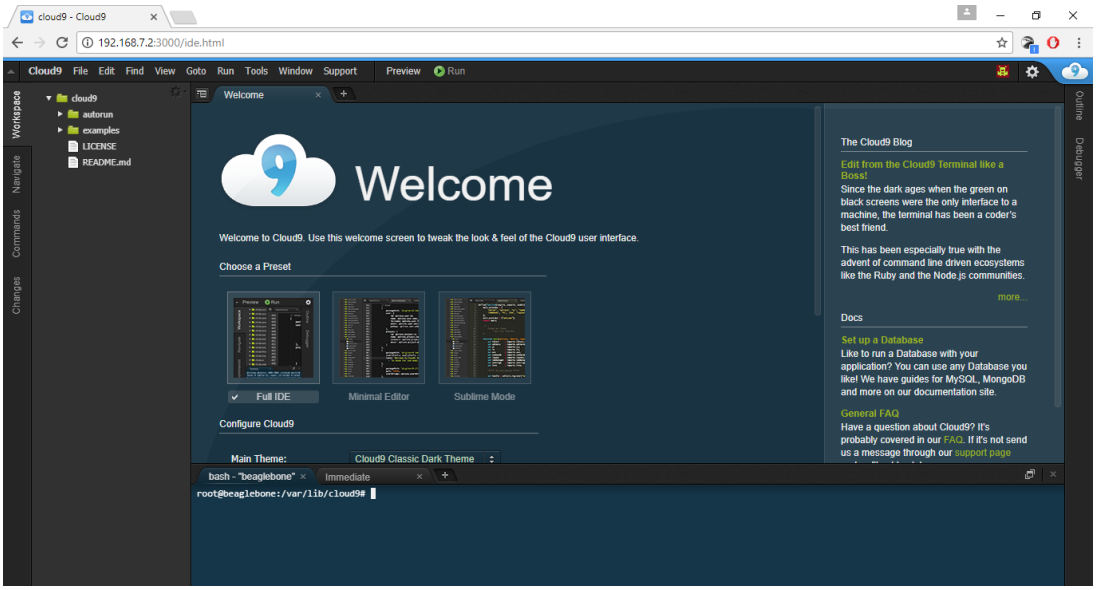


Figura 11.1.0.3 – Entorno de desarrollo integrado Cloud9

Como podemos observar en la parte inferior del navegador, leemos en una pestaña *bash-"beaglebone"*; esa es la consola conectada con la terminal del sistema operativo Debian que se está ejecutando en la BBB, de forma que cualquier comando que introduzcamos ahí se ejecutará instantáneamente en la placa. Si ejecutamos ahí las dos instrucciones, después de un periodo de tiempo de unos 20 minutos, la BBB estará flasheada y el sistema operativo instalado.

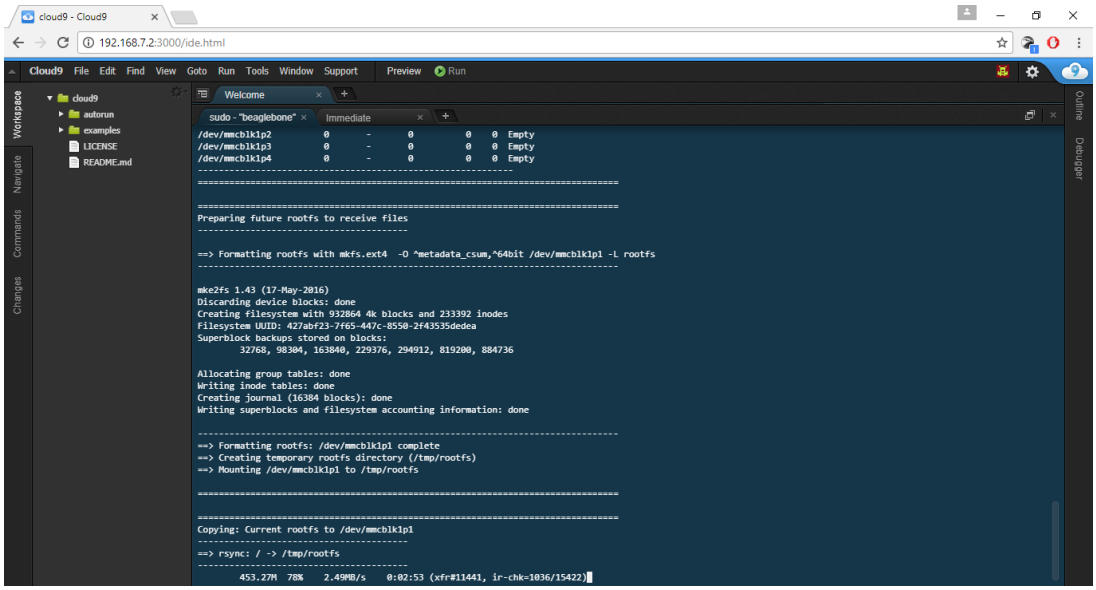


Figura 11.1.0.4 – Proceso de flasheo de la Beaglebone Black

11.2. Cloud9

En esta sección se explicará lo mas básico del entorno de programación Cloud9, así como los pasos para ejecutar correctamente el código que contiene los algoritmos de control.

11.2.1. Manejo básico del entorno Cloud9

Una vez conectada la Beaglebone Black a un PC, y después de introducir en la barra de direcciones del navegador *192.168.7.2:3000*, aparecerá una interfaz como la de la figura 11.2.1.1. En esta, se debe hacer *click* con el botón derecho sobre la carpeta cloud9 y crear una nueva carpeta dentro de esta, en la cual guardaremos los archivos del proyecto.

Dentro de la carpeta creada anteriormente, se puede generar un fichero con extensión *.py* como se ve en la figura 11.2.1.2. Una vez creado, se debe observar que el compilador esté configurado correctamente. Esto lo podemos comprobar en el recuadro rojo de la esquina inferior derecha de la imagen 11.2.1.4, donde vemos que, como debe ser en este caso, esta seleccionado Python. Una vez escrito el código, se ejecuta el programa en el botón *Run*, y aparecerá la salida de este en la parte inferior, en la consola.

11.2.2. Manejo del proyecto en el entorno Cloud9

Una vez sabemos como movernos por el entorno de programación Cloud9, resulta sencillo ejecutar el código que contiene el algoritmo de control PID que estabiliza el sistema objeto.

En primer lugar, debemos abrir el código *RollPitch.py* que se encuentra dentro de la carpeta *TFG*. Una vez abierto, veremos algo similar a lo que se muestra en la figura 11.2.2.1. Aquí, debemos pulsar sobre *Run*, y aparecerá un mensaje en la consola 11.2.2.3 (en rojo), que nos

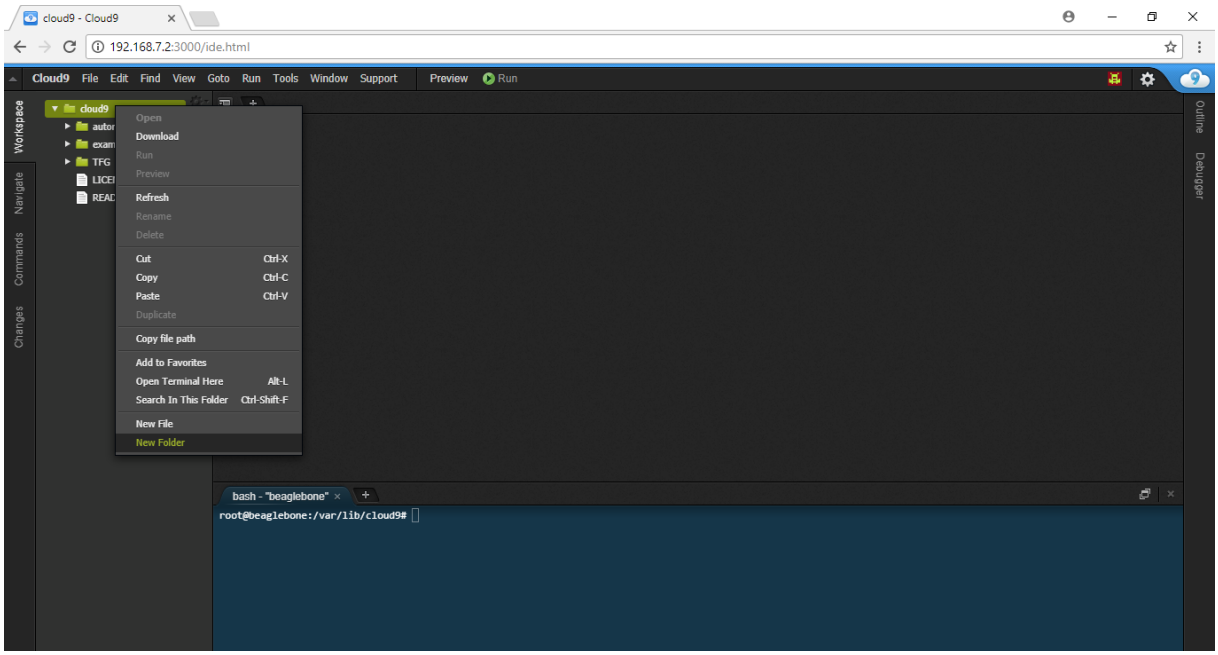


Figura 11.2.1.1 – Creación de una carpeta

pedirá que activemos la alimentación, como vemos en la imagen 11.2.2.2 y pulsemos *Enter*; lo hacemos y el código se ejecutará automáticamente.

En caso de que queramos parar la ejecución del programa, se desconecta la fuente de alimentación y posteriormente se pulsa sobre el botón *Stop* en la interfaz de Cloud9.

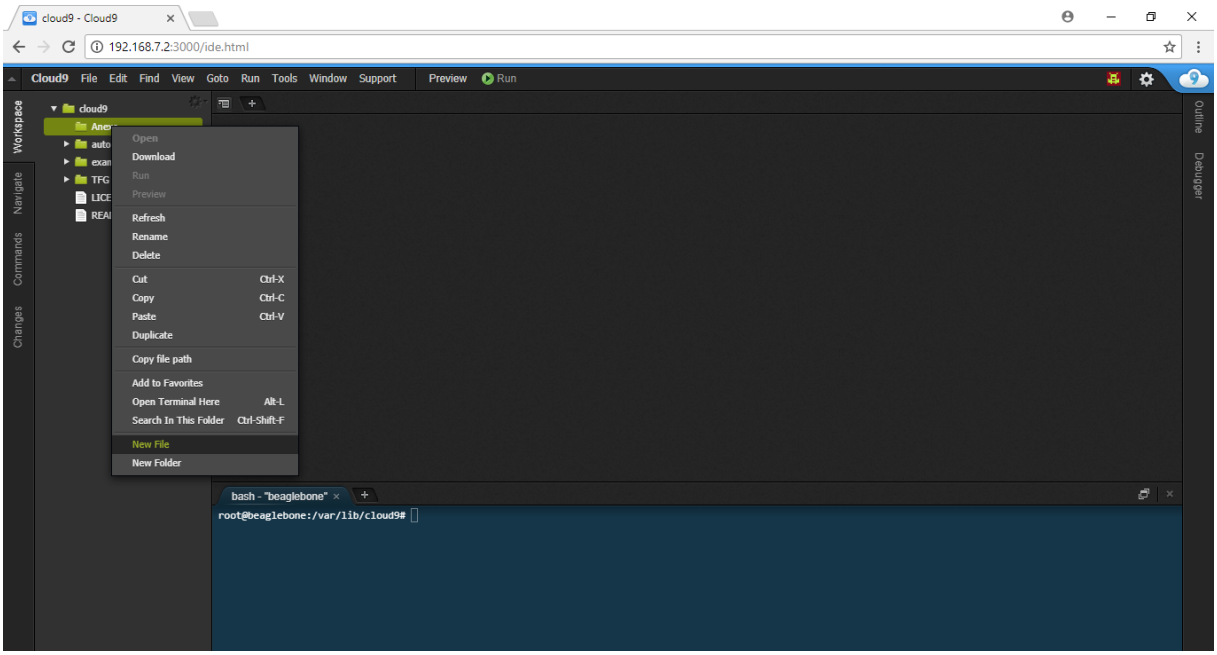


Figura 11.2.1.2 – Creación de un fichero

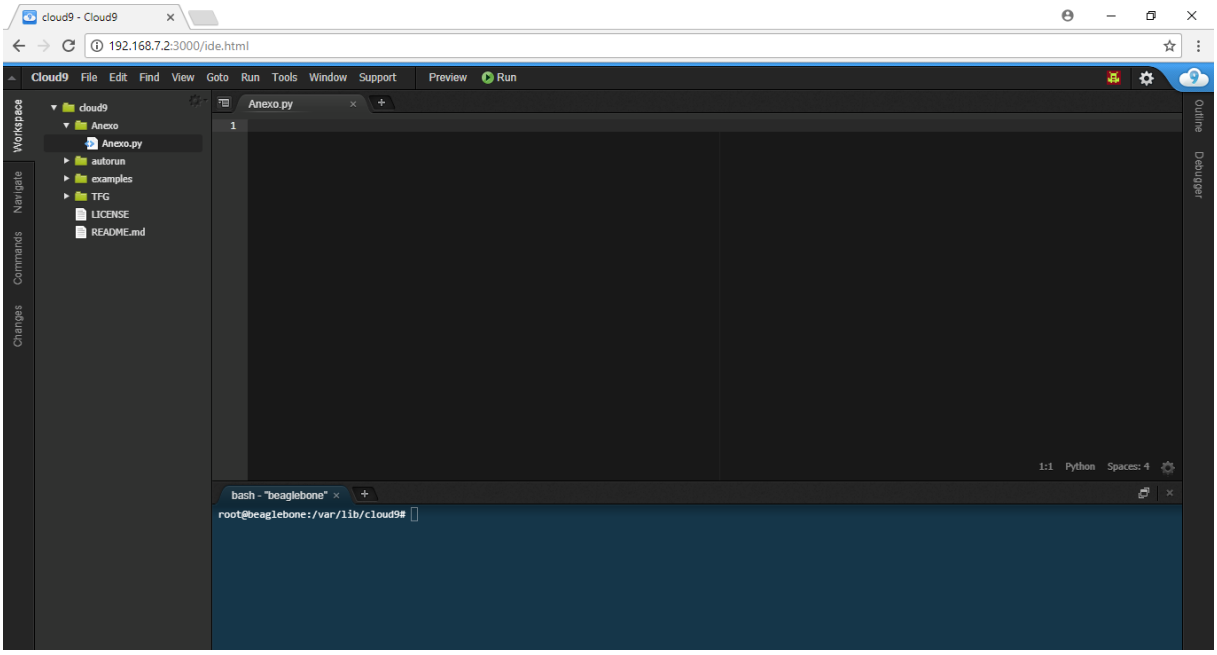


Figura 11.2.1.3 – Vista de un fichero abierto

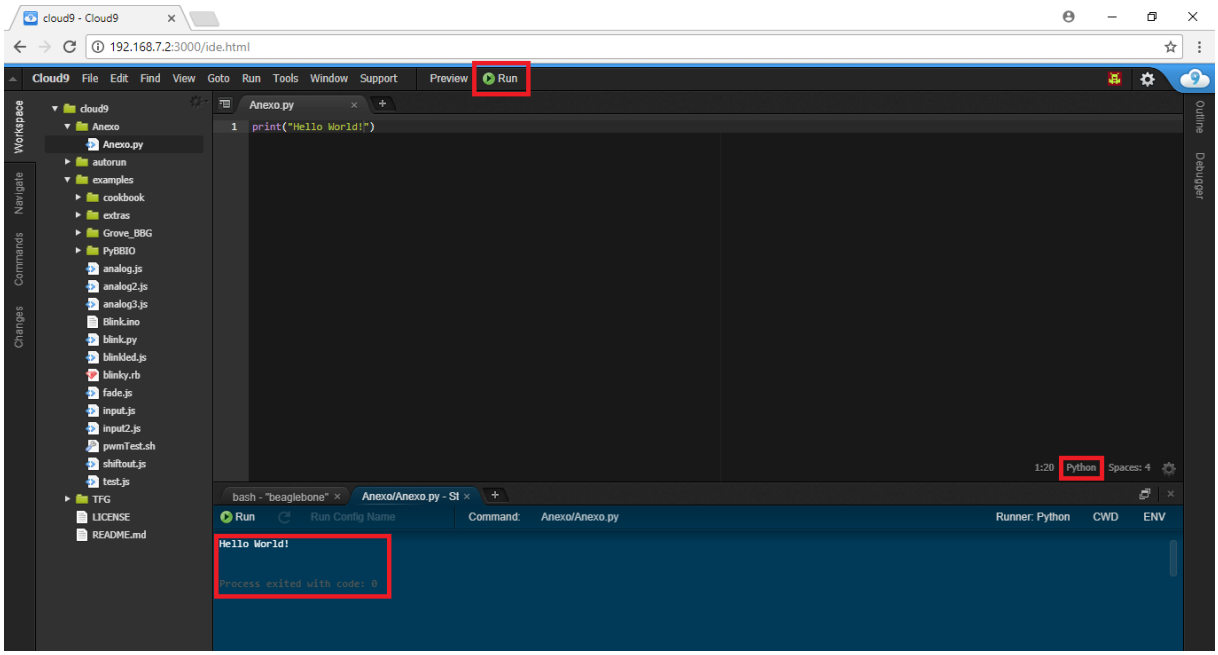


Figura 11.2.1.4 – Ejecución del programa

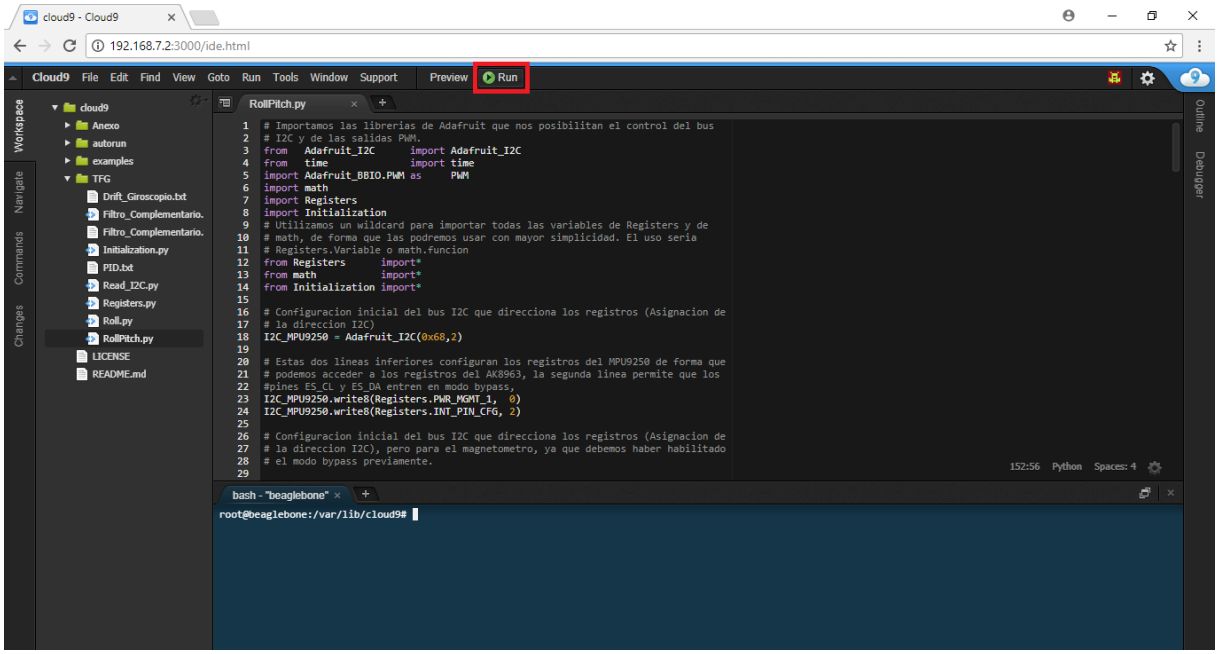


Figura 11.2.2.1 – Vista del código que se desea ejecutar

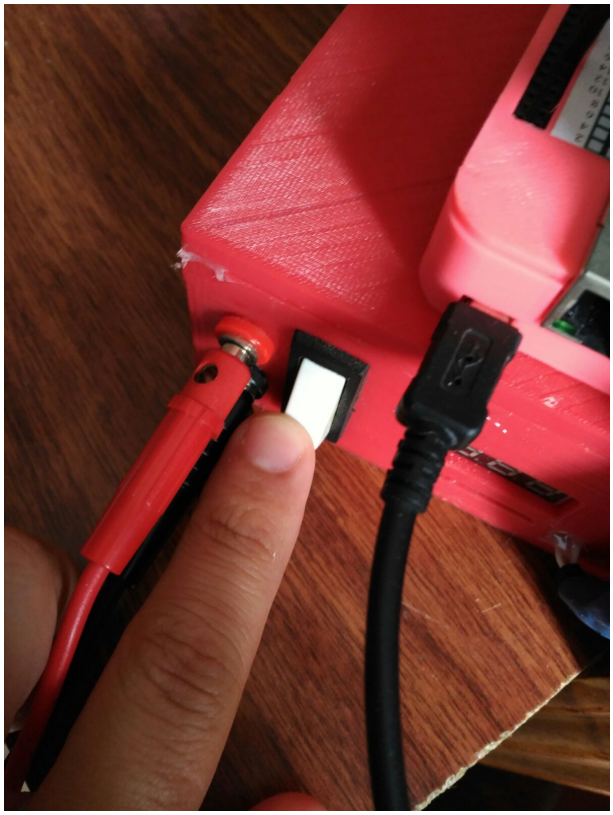


Figura 11.2.2.2 – Activación de la fuente de alimentación

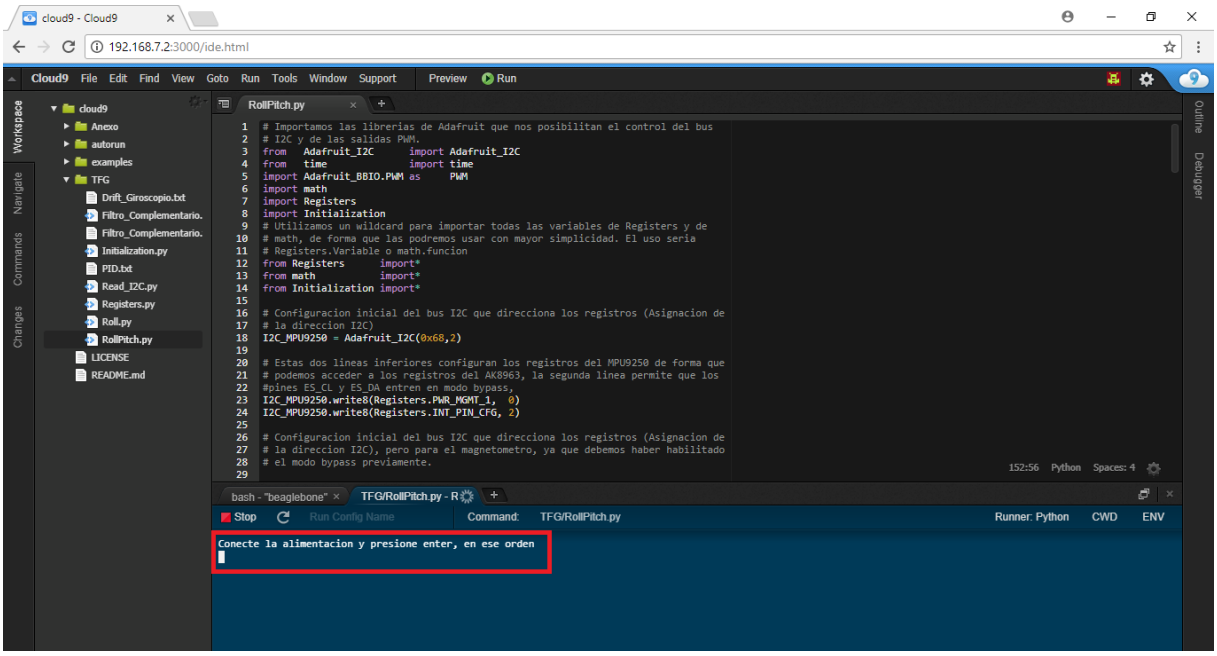


Figura 11.2.2.3 – Paso previo a la ejecución del algoritmo de control

11.3. Impresión de las piezas en 3D

Como se ha mencionado con anterioridad, se ha utilizado la impresora 3D BQ Hephestos Prusa i3. Una vez están diseñadas las piezas en el software NX, exportamos estas como archivos con la extensión *.stl*, como vemos en la captura 11.3.0.1. STL es un tipo de archivo de diseño CAD que define la geometría de figuras 3D, prescindiendo, a diferencia de otros tipos de archivos CAD, de características como el color, las propiedades físicas u texturas.

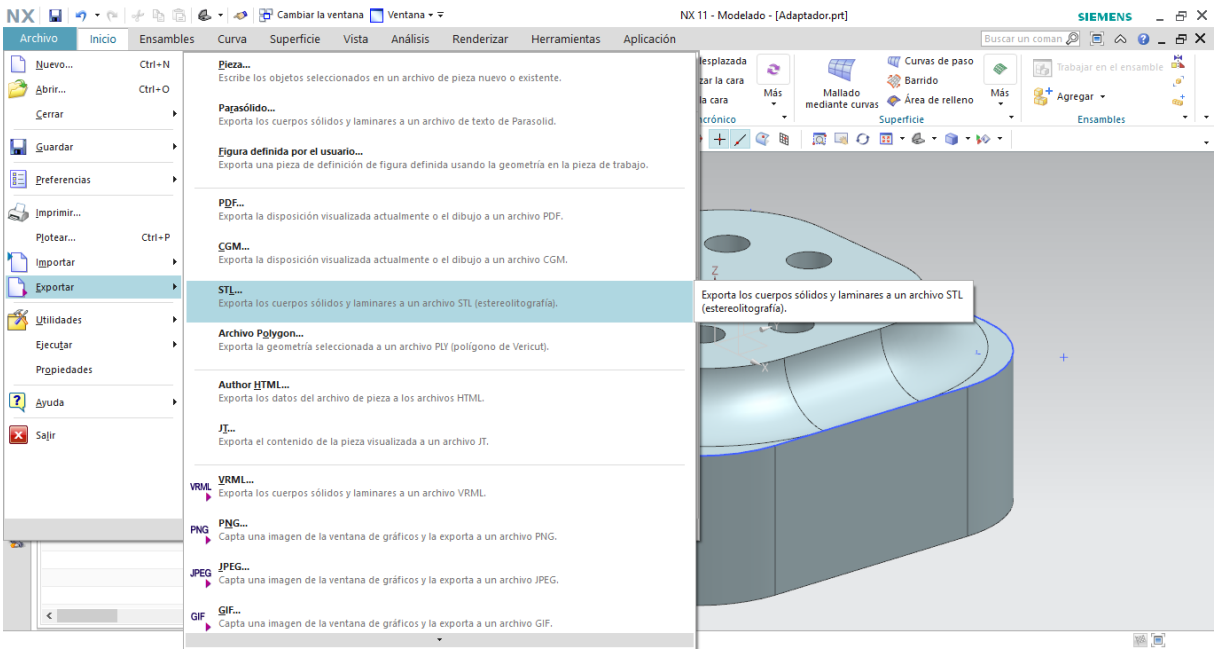


Figura 11.3.0.1 – Submenú que exporta la figura actual como archivo STL

El siguiente paso es convertir el fichero STL en algo interpretable por la impresora 3D, para ello, utilizaremos el software Cura, que dispone de una amplia biblioteca de impresoras 3D (Esto es importante, ya que cada impresora tiene unas determinadas características técnicas). Una vez hemos instalado el software de CURA, lo abrimos, y vemos la pantalla de inicio del programa 11.3.0.2. En este punto, debemos importar el archivo STL que hemos creado anteriormente, asegurarnos de que está seleccionada la impresora correcta, seleccionar la calidad de la impresión en el desplegable *Perfil*, el material, en nuestro caso PLA, y hacer click sobre *Guardar en unidad extraible* o *Guardar en archivo*. Esto último nos generará un archivo con extensión *.gcode*, que guardaremos en una tarjeta SD compatible con la impresora, esta tarjeta se inserta en la impresora, como vemos en la fotografía

11.4. Software BLHeli y Adaptador

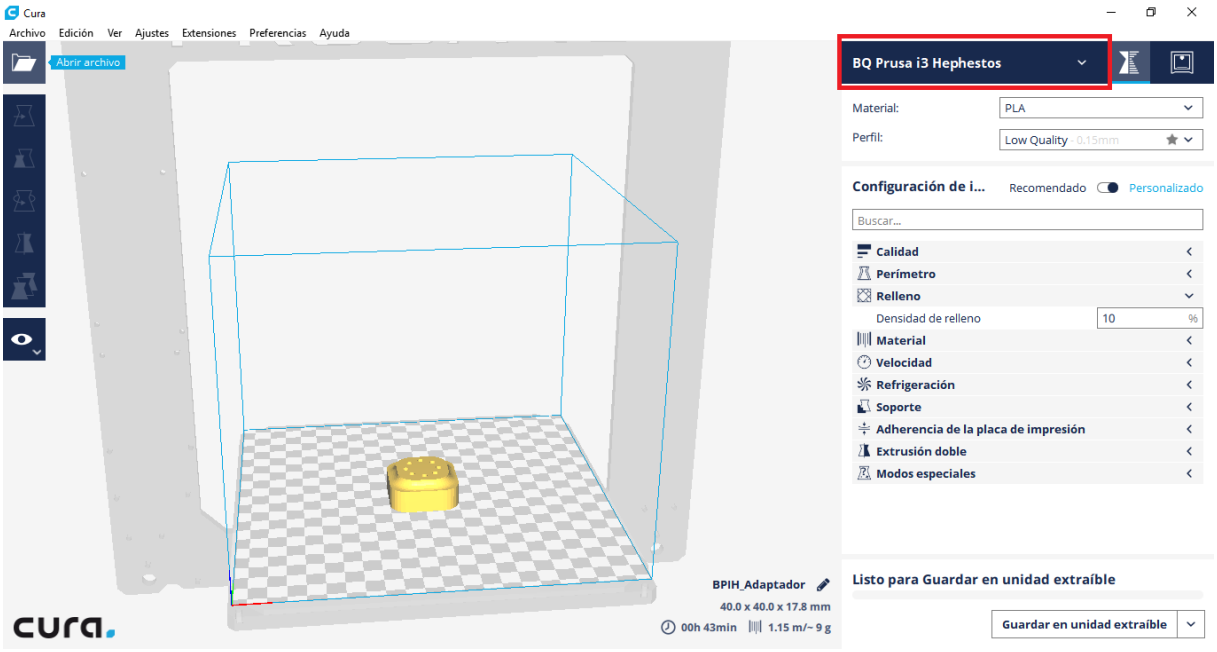


Figura 11.3.0.2 – Vista del software CURA

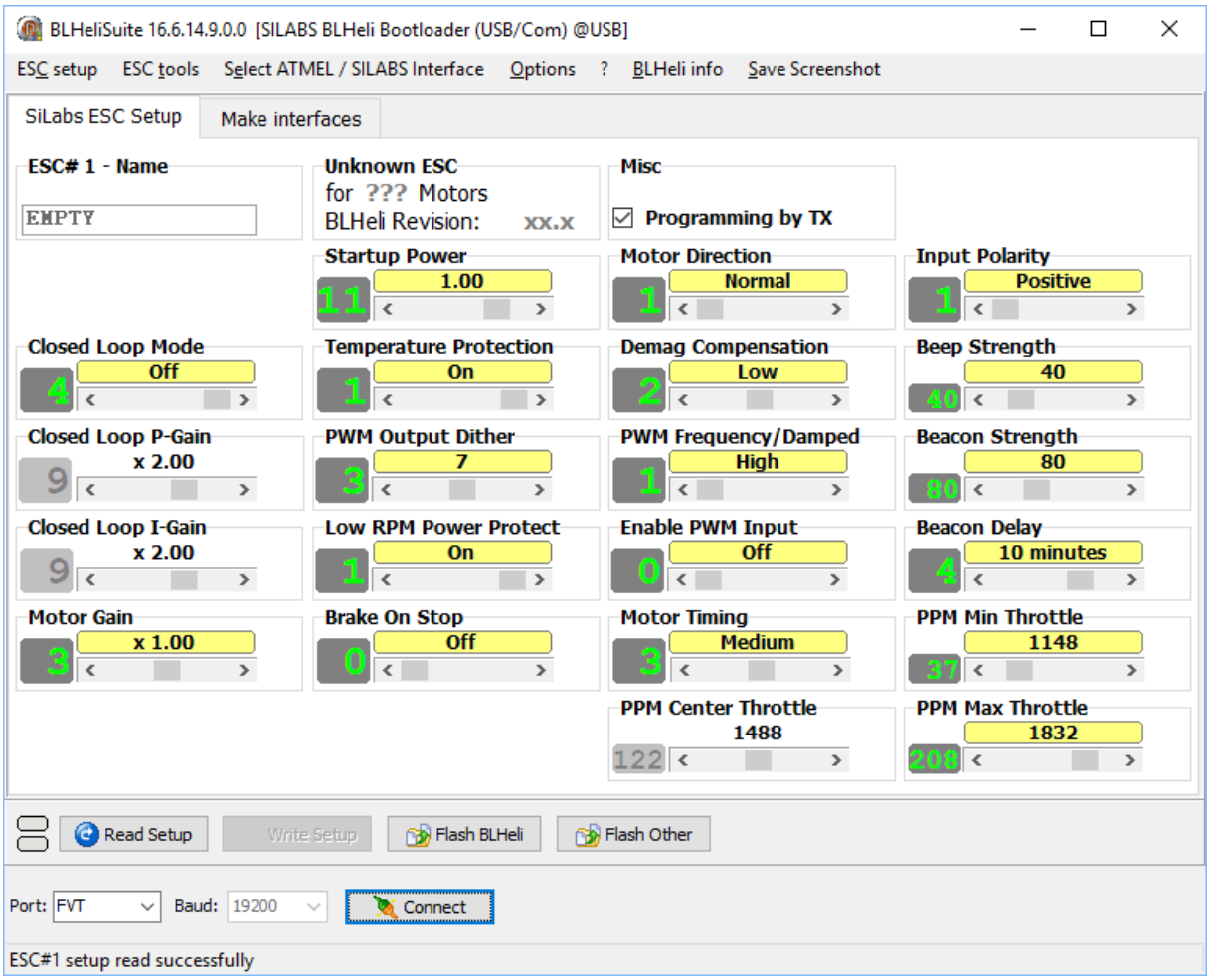


Figura 11.4.0.1 – Vista de la ventana principal del software

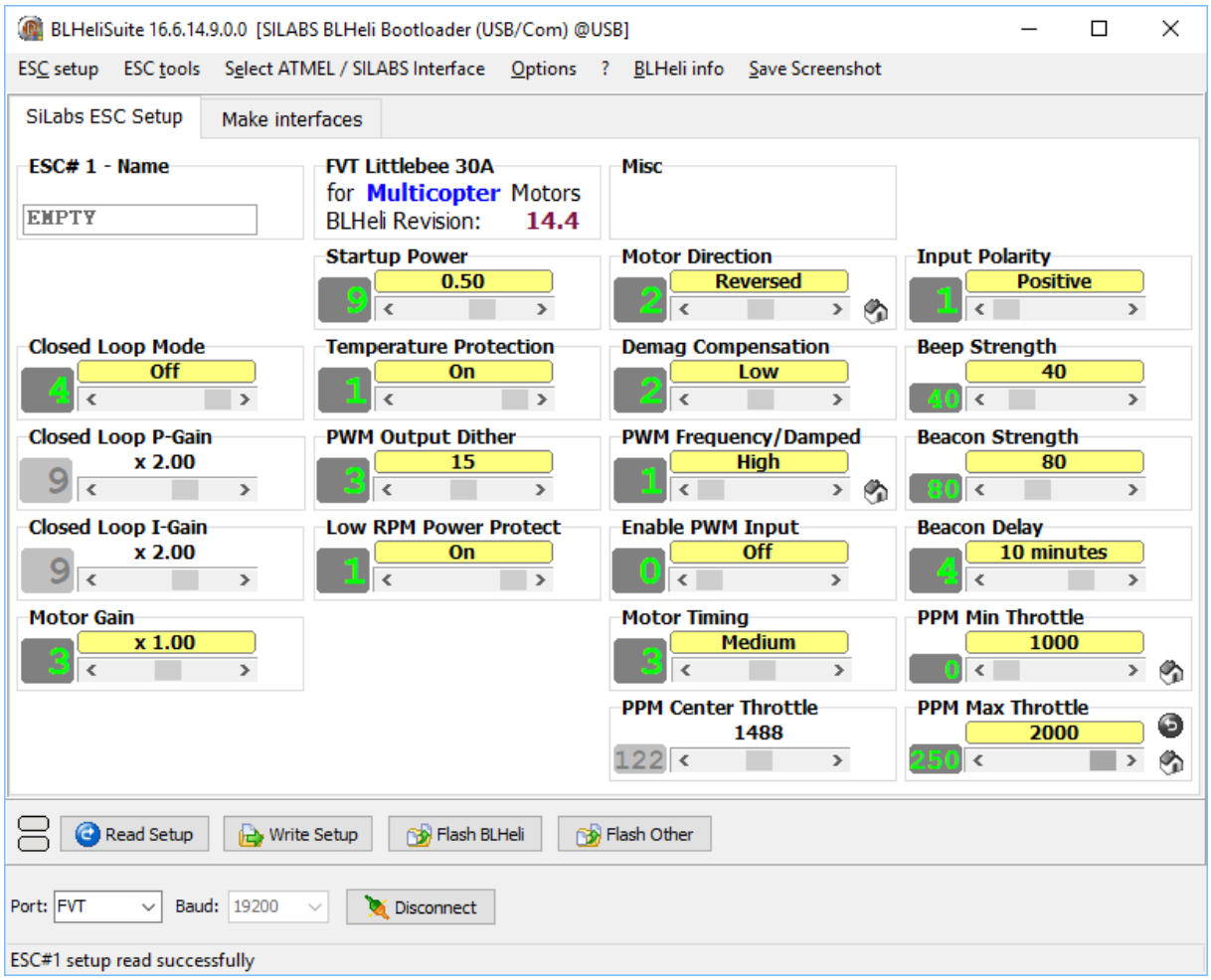


Figura 11.4.0.2 – Pantalla de edición de parámetros

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

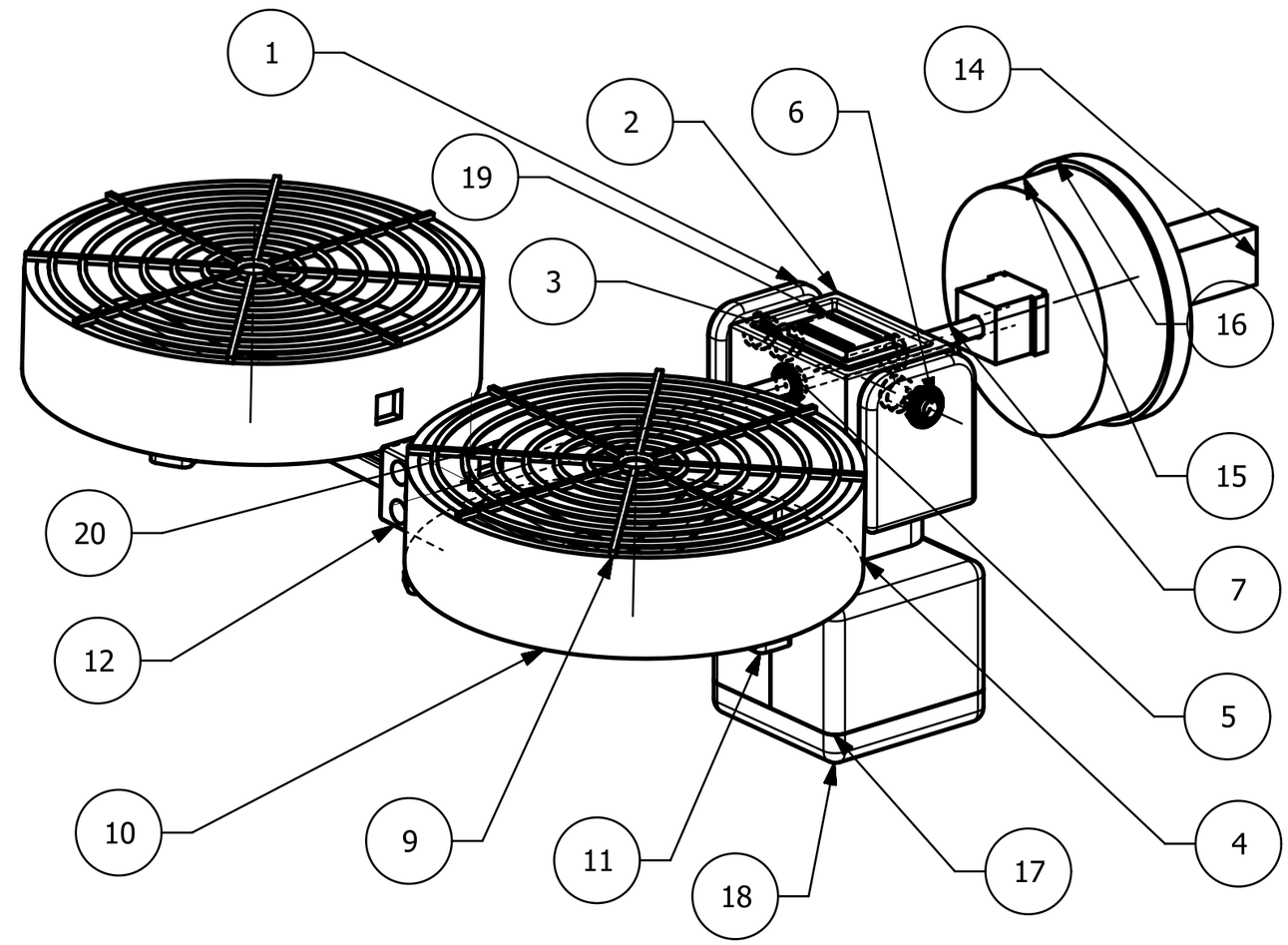
FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice de planos

Nº	NOMBRE DE LA PARTE	CTD
1	CAPA 2 ROTULA	1
2	CAPA 1 ROTULA	1
3	CAPA 0 ROTULA	1
4	ADAPTADOR	1
5	RODAMIENTO 1/4 "	6
6	VARILLA RECORTADA 0.79" X 1/4 "	2
7	VARILLA 9" X 1/4"	1
8	TORNILLO ESTRELLA 6-32 X 1"	8
9	CABEZA PROTECTOR HELICE	2
10	BASE PROTECTOR HELICE	2
11	BASE HELICE	2
12	SOPORTE HELICES	1
13	ENSAMBLE DEL PROPULSOR (NO ES PIEZA)	1
14	CARRIL CONTRAPESO	1
15	CONTRAPESO	1
16	TAPA CONTRAPESO	1
17	TAPA MOTOR ROTULA	2
18	BASE MOTOR ROTULA	1
19	SOPORTE SENSOR	1
20	ACOPLE DE TORNILLO 1/4"	1



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

PLANO GENERAL

FECHA: SEP 2017

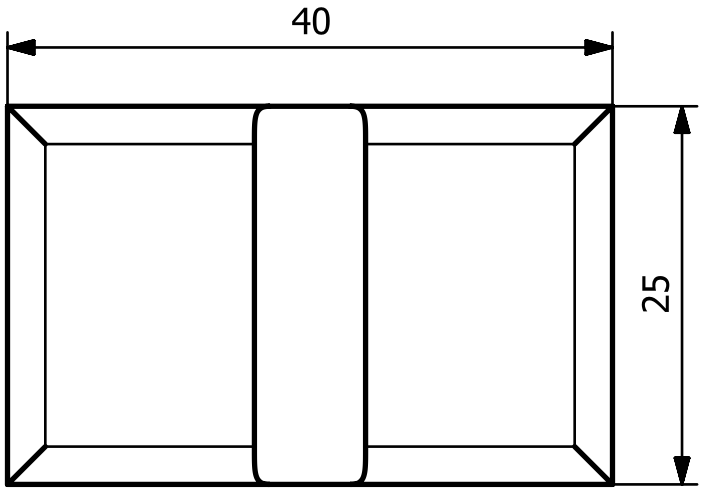
ESCALA: 1:2

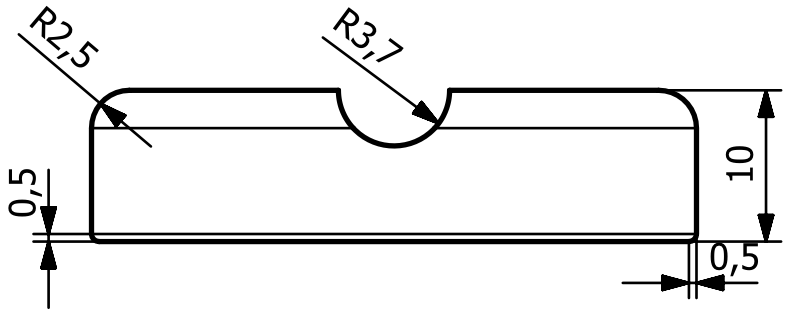
AUTOR: CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 01

A3





UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

CAPA 0 RÓTULA

FECHA: SEP 2017

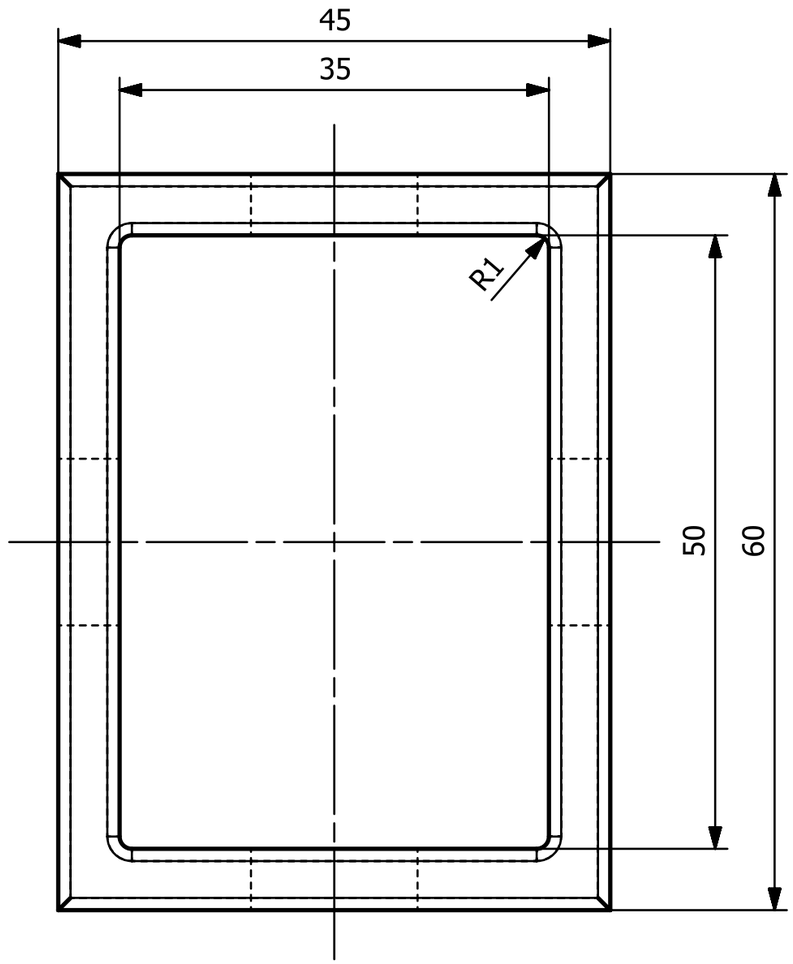
ESCALA: 2:1

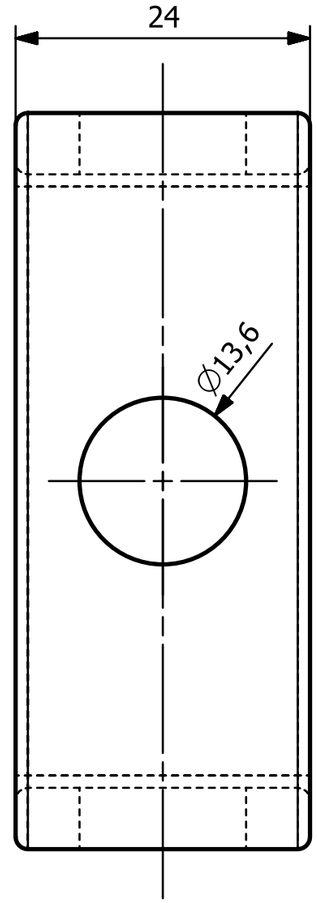
AUTOR:

CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 02





UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

CAPA 1 RÓTULA

FECHA: SEP 2017

ESCALA: 2:1

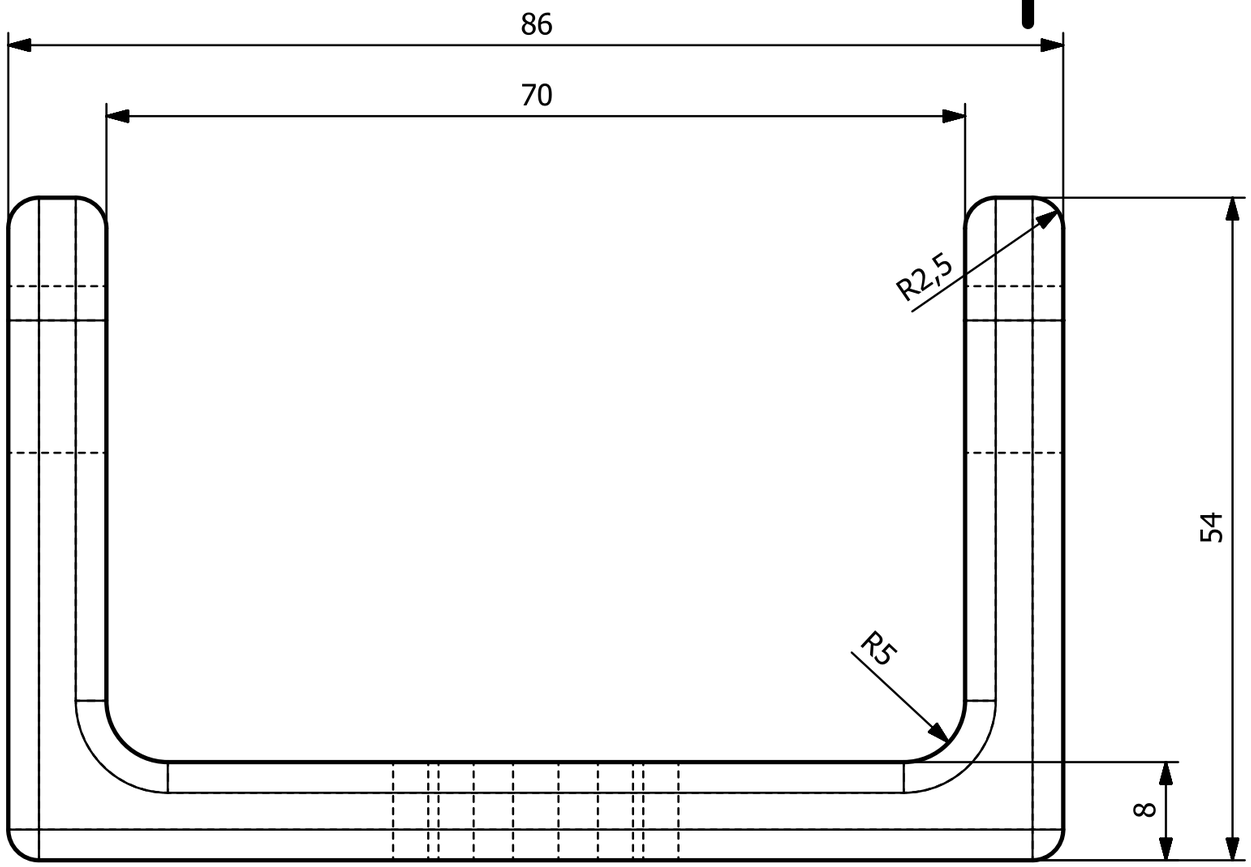
AUTOR:

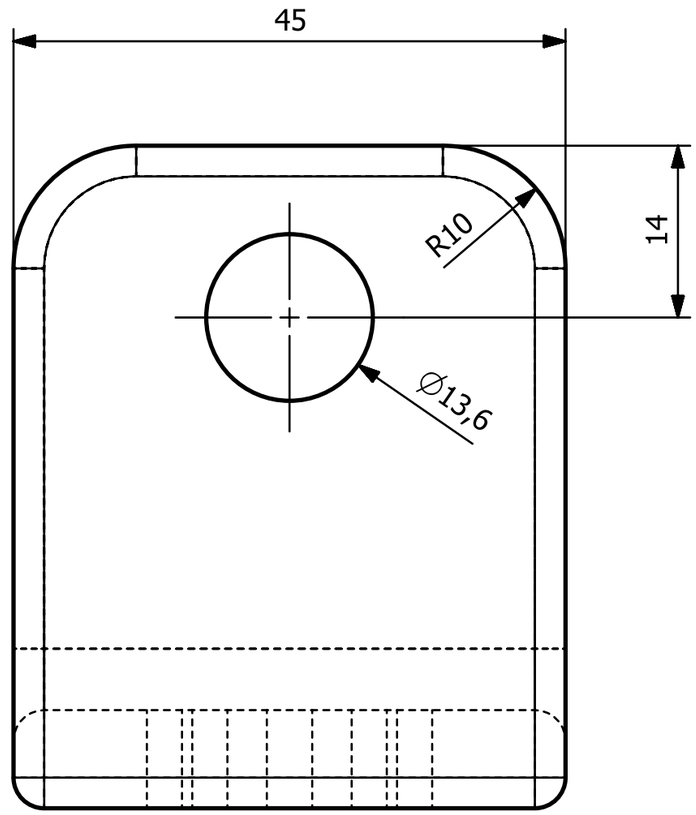
CONSTANTINO BELLO CORBEIRA

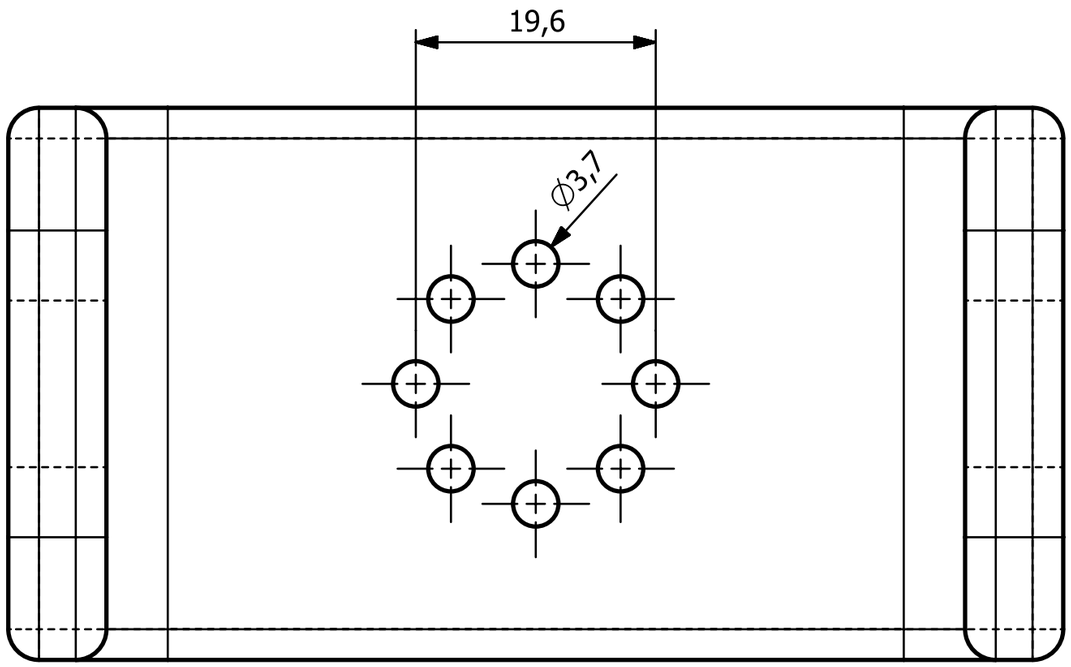
FIRMA:

PLANO Nº: 03

A3







UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

CAPA 2 RÓTULA

FECHA: SEP 2017

ESCALA: 2:1

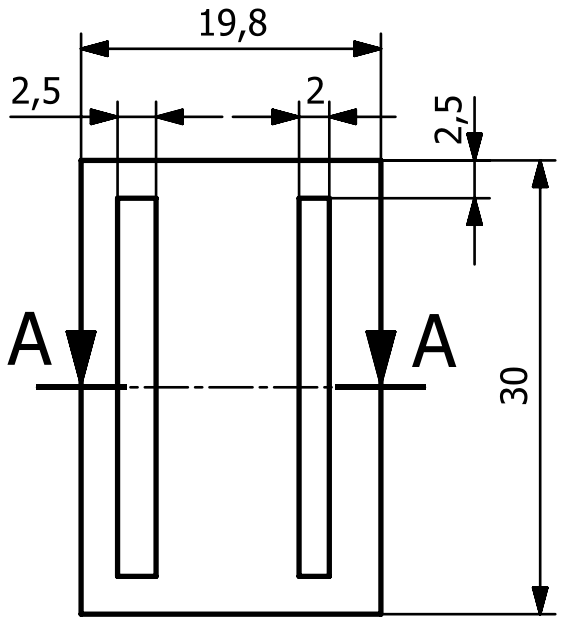
AUTOR:

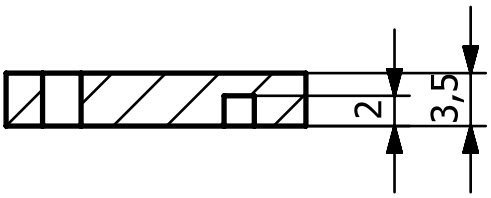
CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 04

A3





SECTION A-A

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

SOPORTE SENSOR

FECHA: SEP 2017

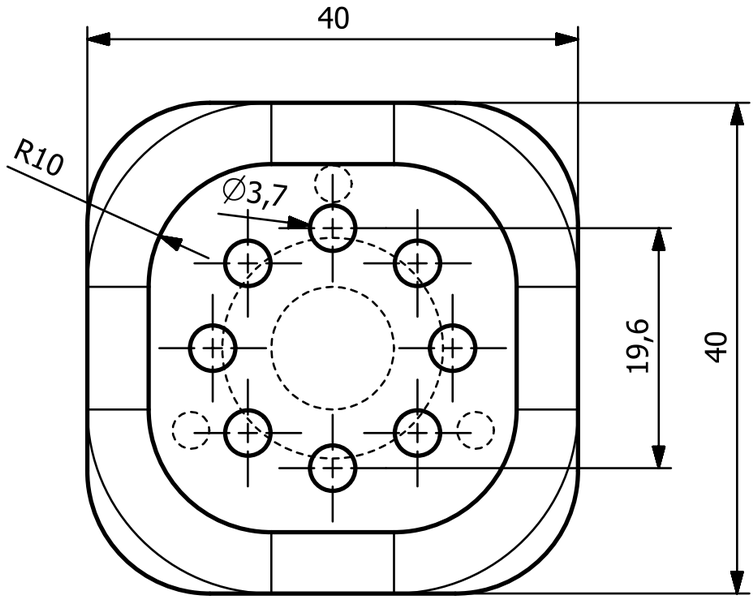
ESCALA: 2:1

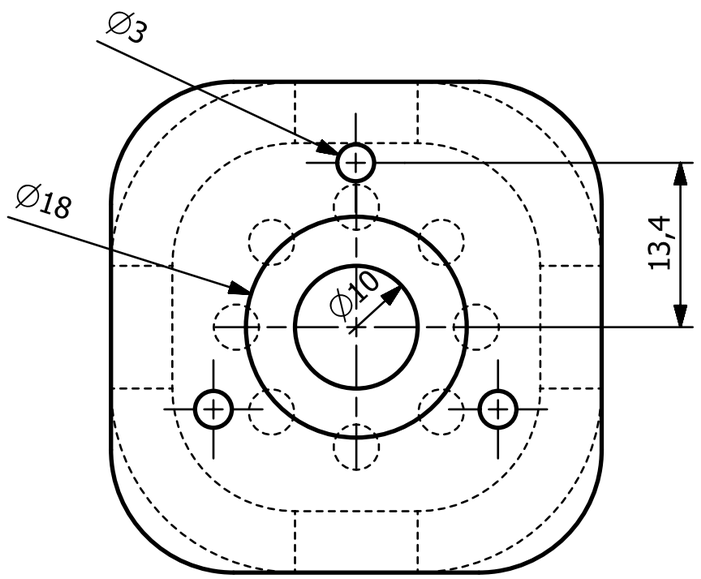
AUTOR:

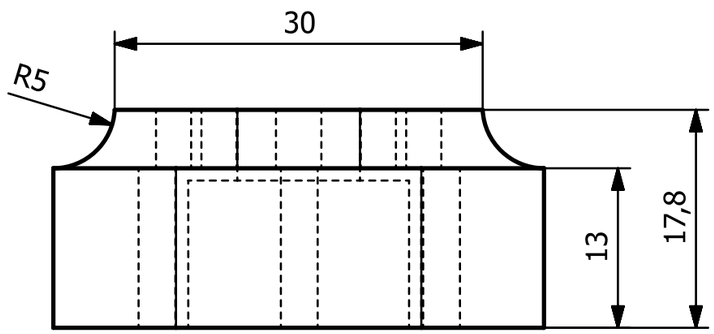
CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 05







UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

ADAPTADOR

FECHA: SEP 2017

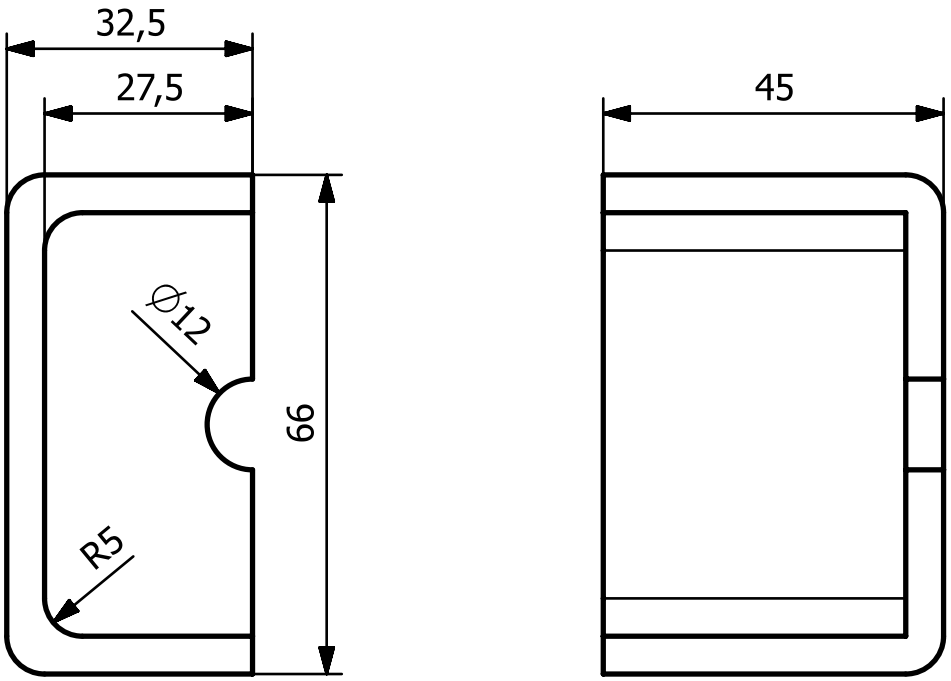
ESCALA: 2:1

AUTOR: CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 06

A3



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

CUBIERTA MOTOR RÓTULA

FECHA: SEP 2017

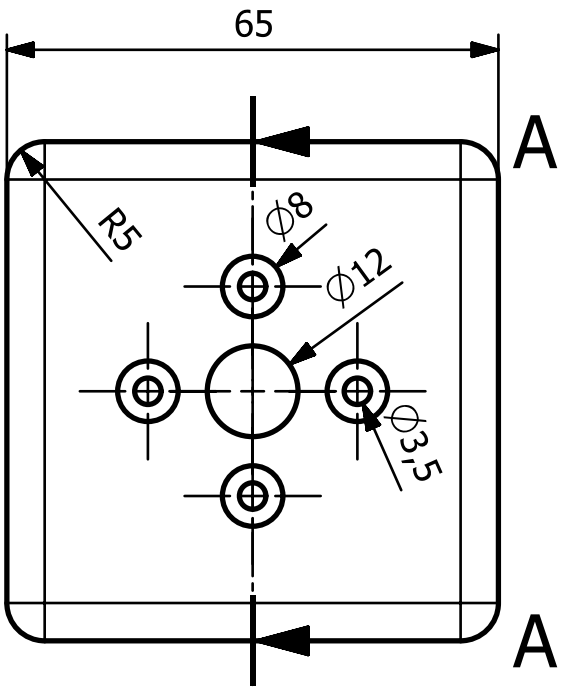
ESCALA: 1:1

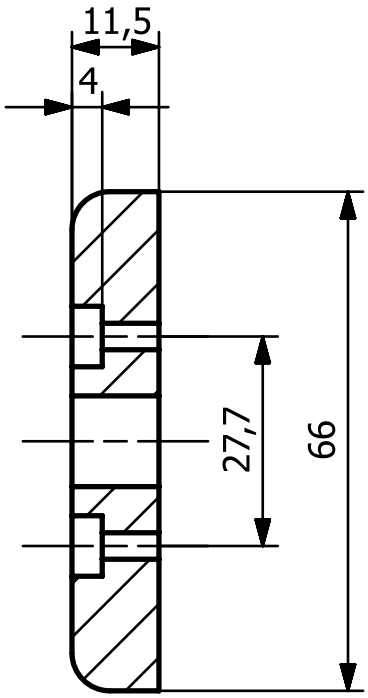
AUTOR:

CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 07





SECTION A-A

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

BASE MOTOR RÓTULA

FECHA: SEP 2017

ESCALA: 1:1

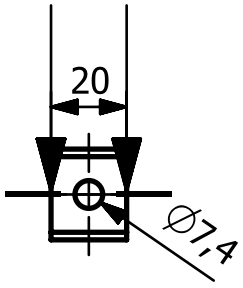
AUTOR:

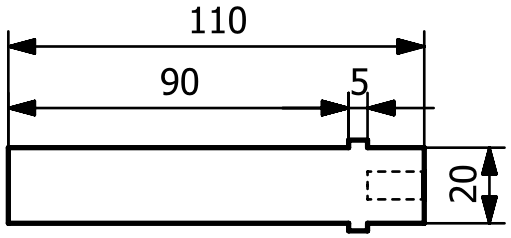
CONSTANTINO BELLO CORBEIRA

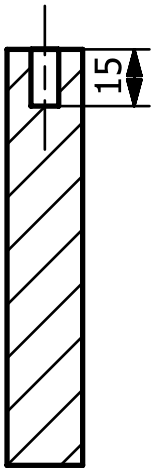
FIRMA:

PLANO Nº: 08

A A







SECTION A-A

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

CARRIL CONTRAPESO

FECHA: SEP 2017

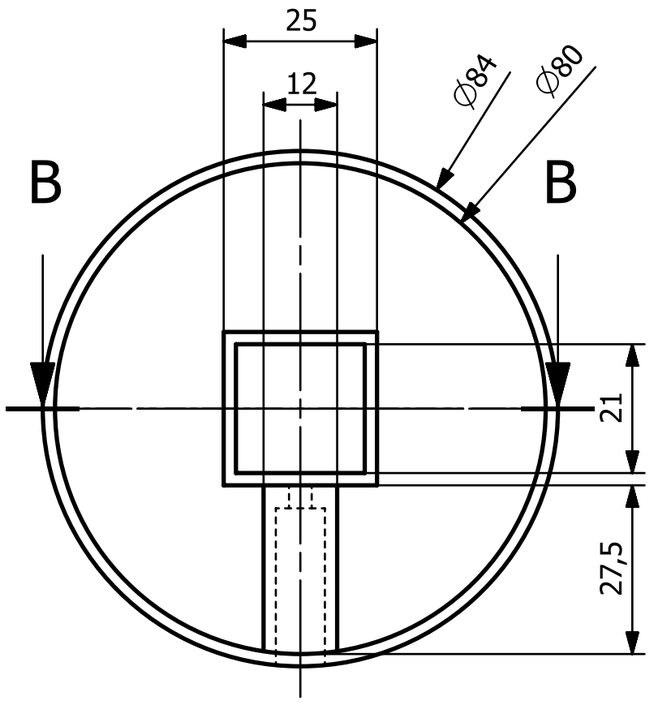
ESCALA: 1:2

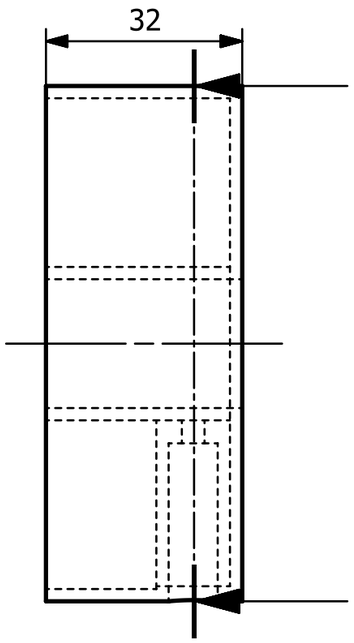
AUTOR:

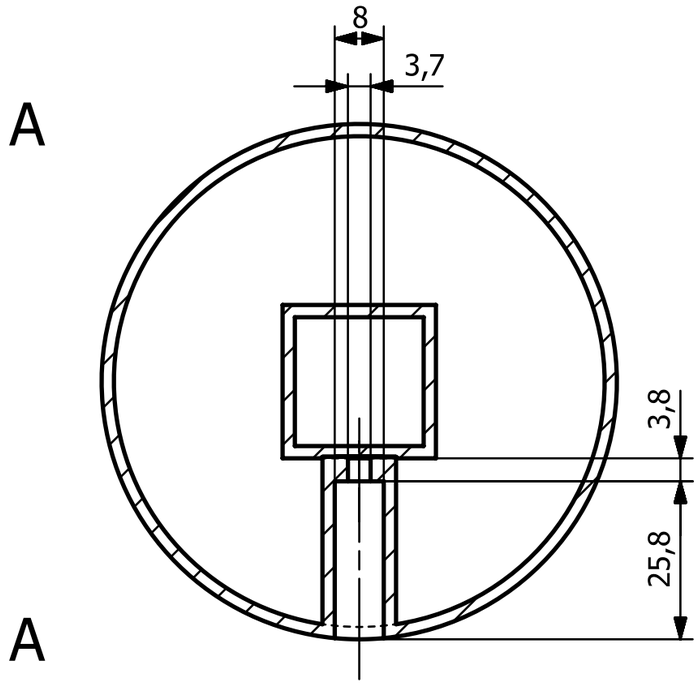
CONSTANTINO BELLO CORBEIRA

FIRMA:

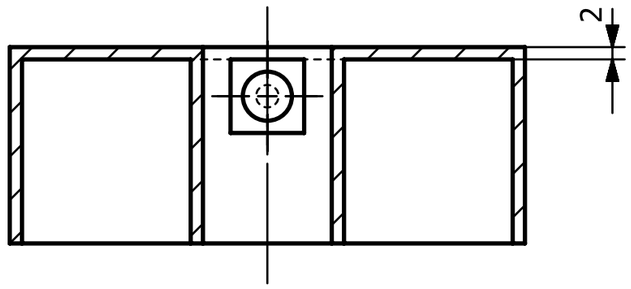
PLANO Nº: 09





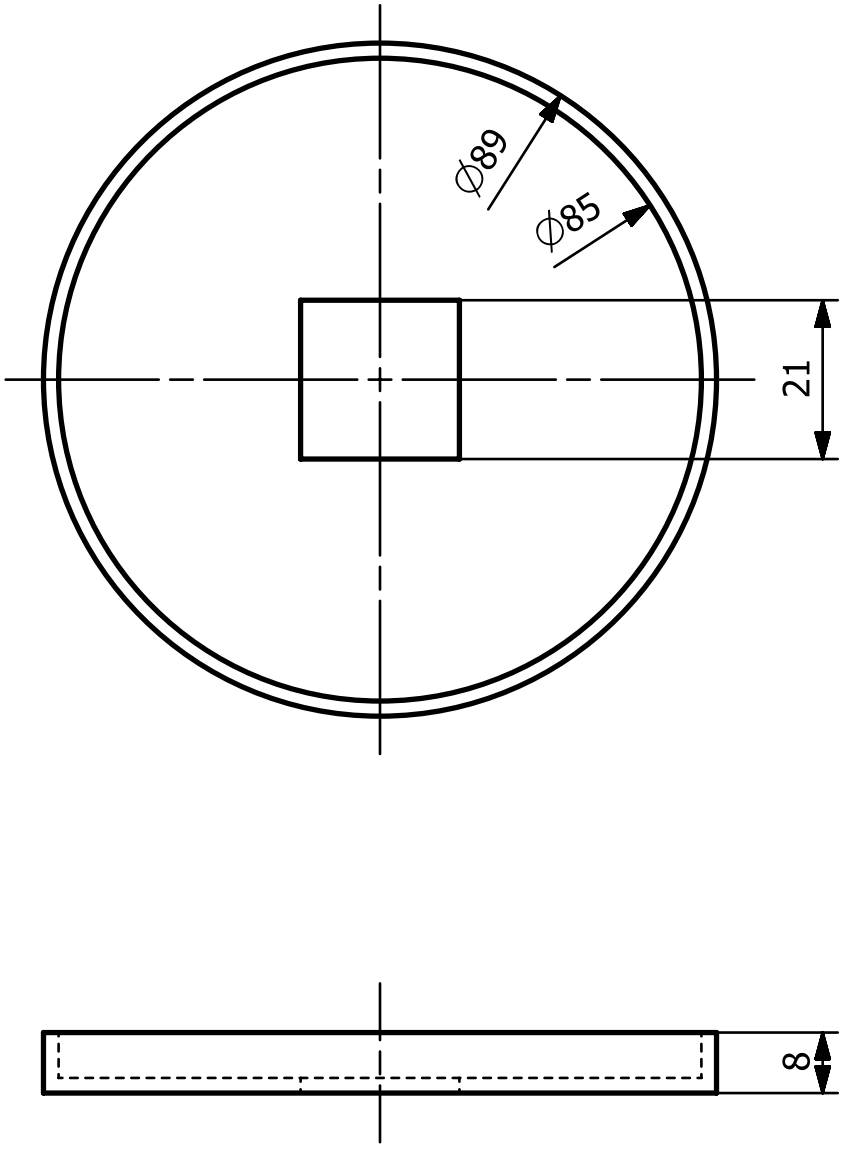


SECTION A-A



SECTION B-B

UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA		TFG Nº: 770G01A120
TÍTULO DEL TFG: ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO		
TÍTULO DEL PLANO: CONTRAPESO		FECHA: SEP 2017
		ESCALA: 1:1
AUTOR: CONSTANTINO BELLO CORBEIRA	FIRMA:	PLANO Nº: 10
		A3



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

TAPA CONTRAPESO

FECHA: SEP 2017

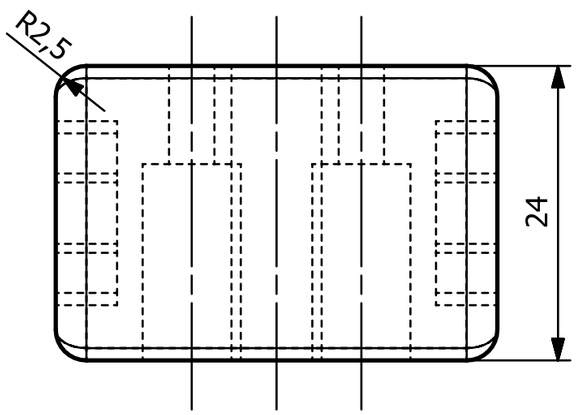
ESCALA: 1:1

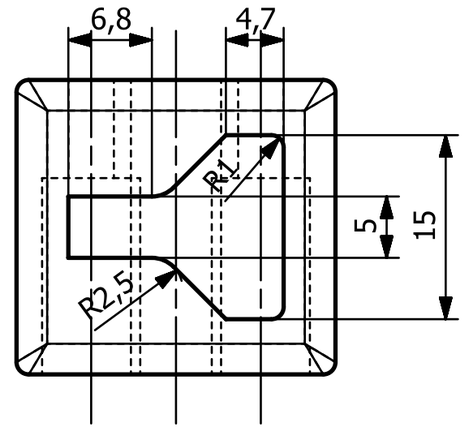
PLANO Nº: 11

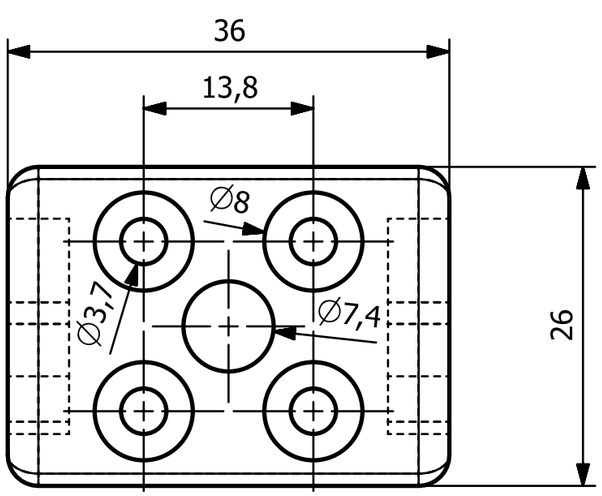
AUTOR:

CONSTANTINO BELLO CORBEIRA

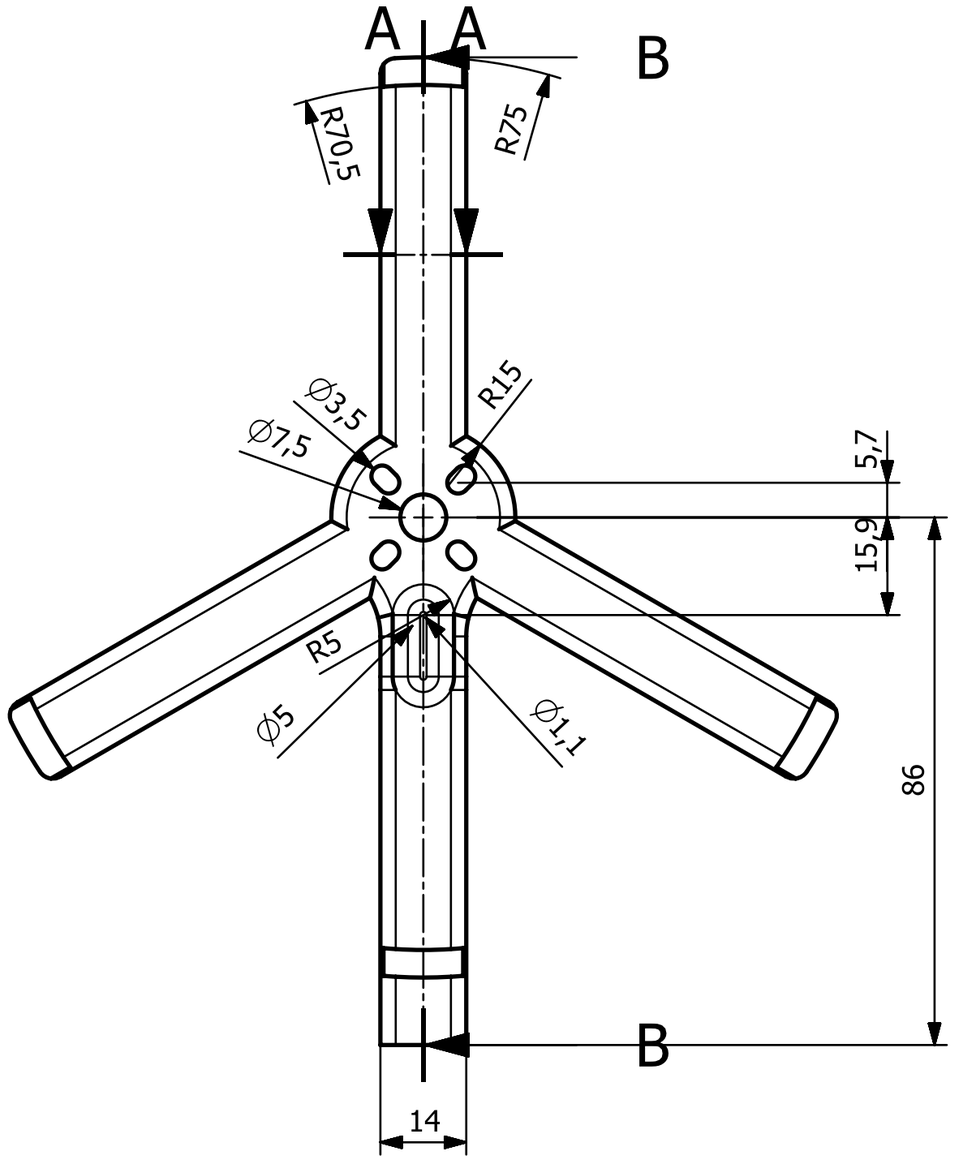
FIRMA:

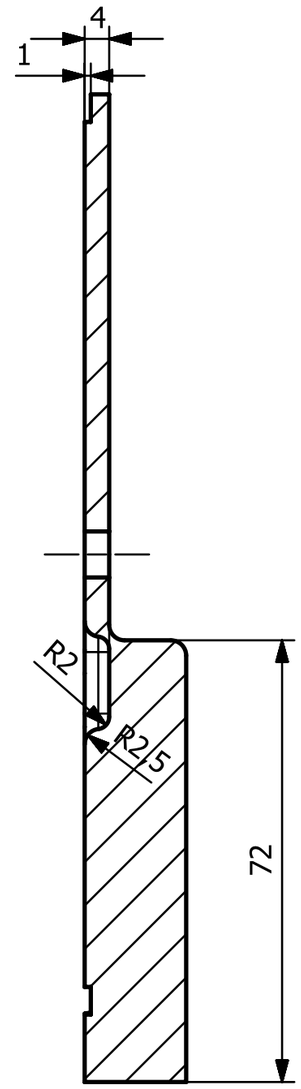




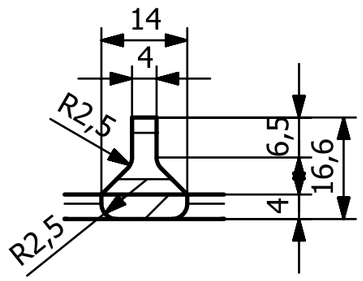


UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A120
TÍTULO DEL TFG: ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO	
TÍTULO DEL PLANO: SOPORTE HÉLICES	FECHA: SEP 2017
AUTOR: CONSTANTINO BELLO CORBEIRA — FIRMA:	ESCALA: 2:1
	PLANO Nº: 12
	A3





SECTION B-B



SECTION A-A

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

BASE HÉLICE

FECHA: SEP 2017

ESCALA: 1:1

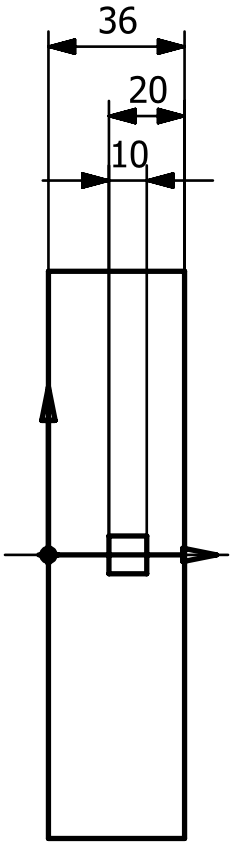
AUTOR:

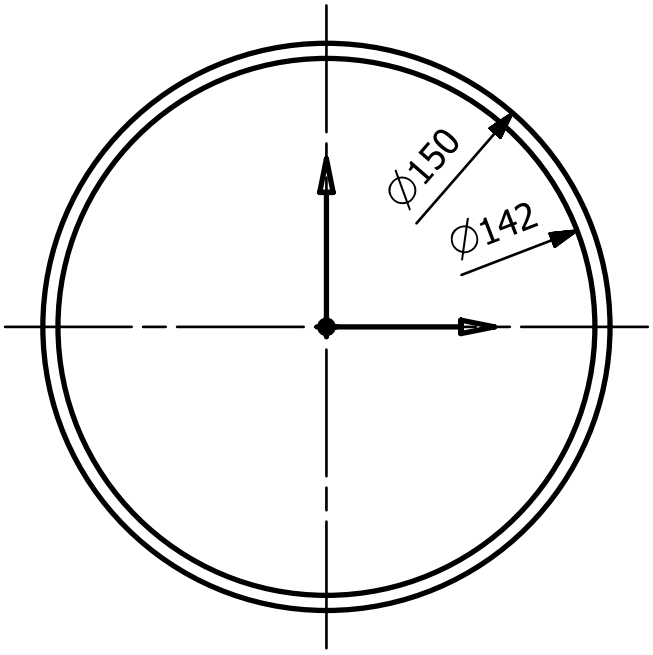
CONSTANTINO BELLO CORBEIRA

FIRMA:

PLANO Nº: 13

A3





UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

BASE PROTECCIÓN HÉLICE

FECHA: SEP 2017

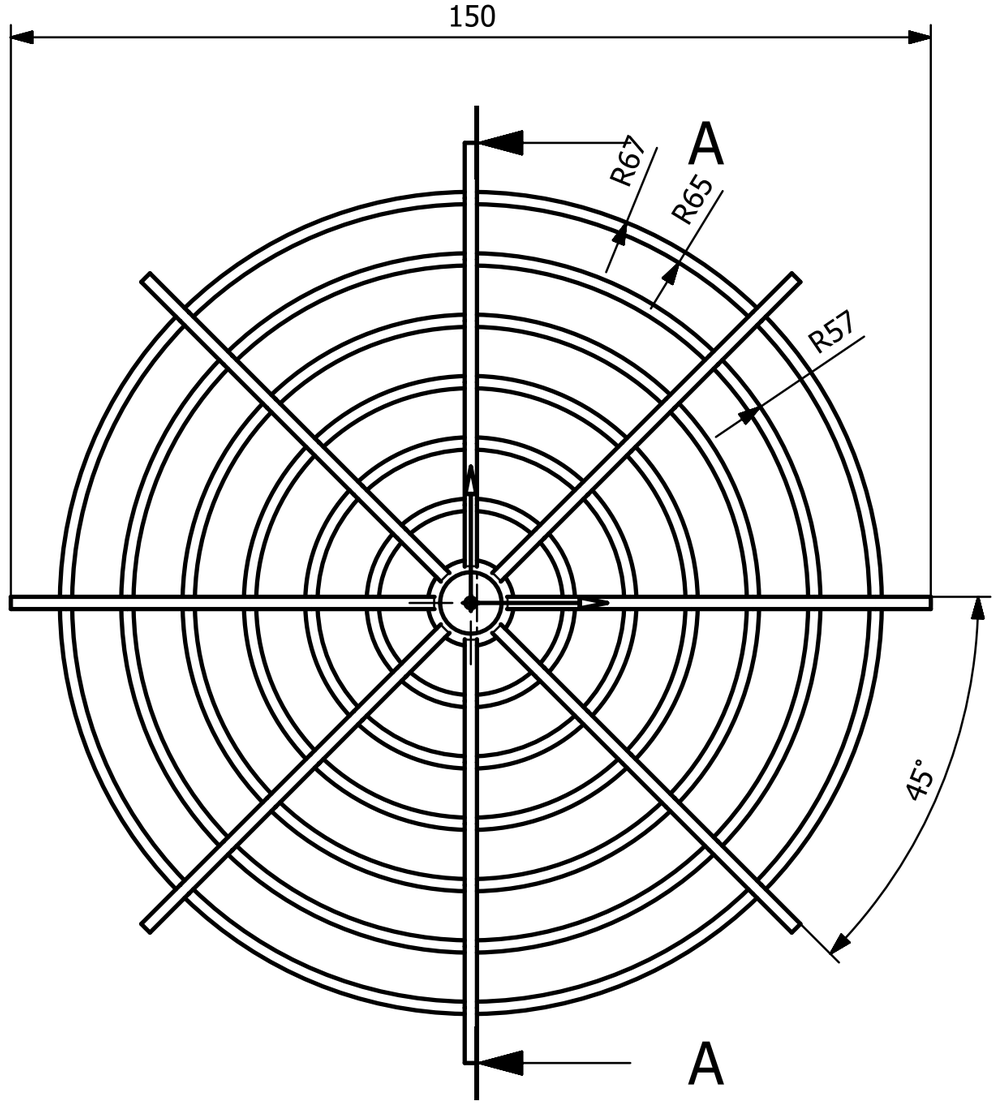
ESCALA: 1:2

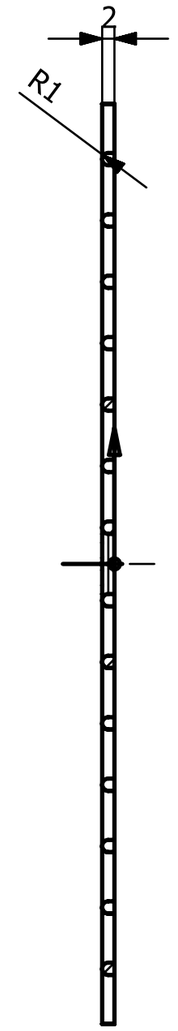
PLANO Nº: 14

AUTOR:

CONSTANTINO BELLO CORBEIRA

FIRMA:





SECTION A-A

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

REJILLA PROTECCIÓN HÉLICE

FECHA: SEP 2017

ESCALA: 1:2

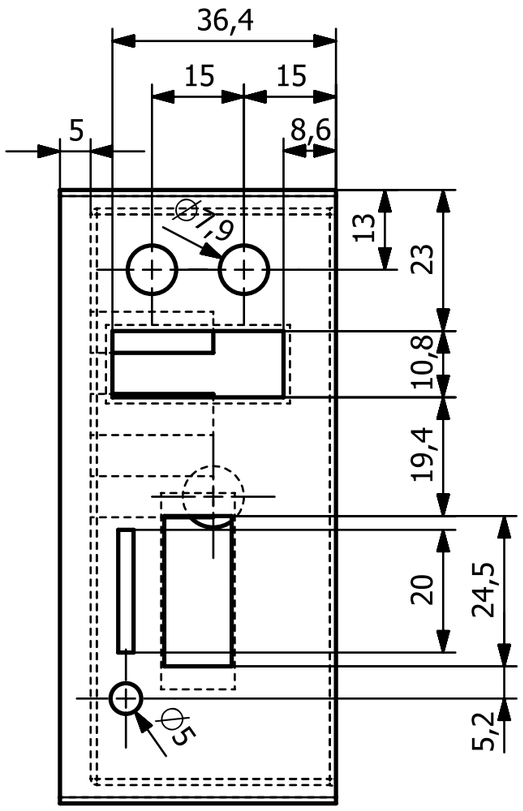
PLANO Nº: 15

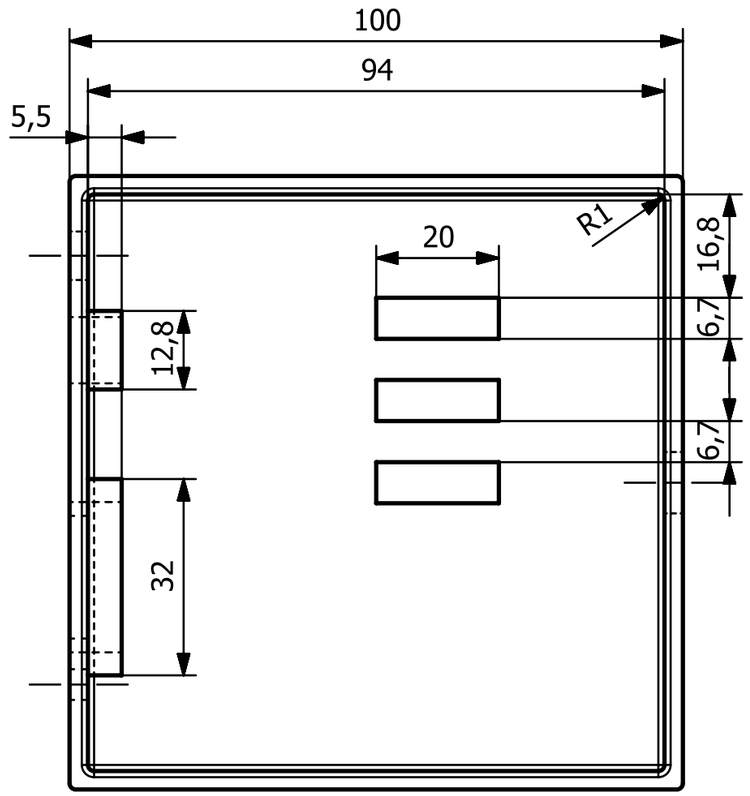
AUTOR:

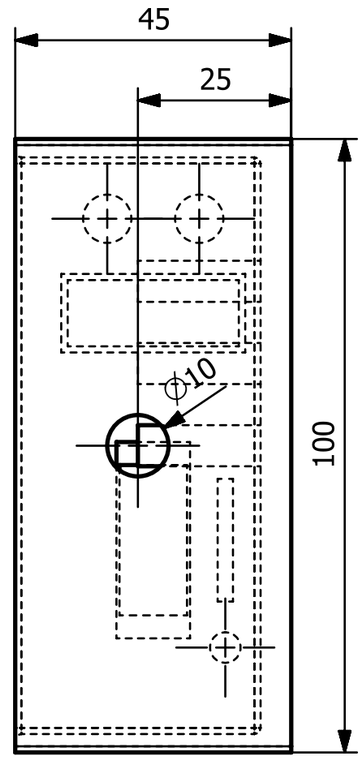
CONSTANTINO BELLO CORBEIRA

FIRMA:

A3







UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

BASE CAJA ELECTRÓNICA

FECHA: SEP 2017

ESCALA: 1:1

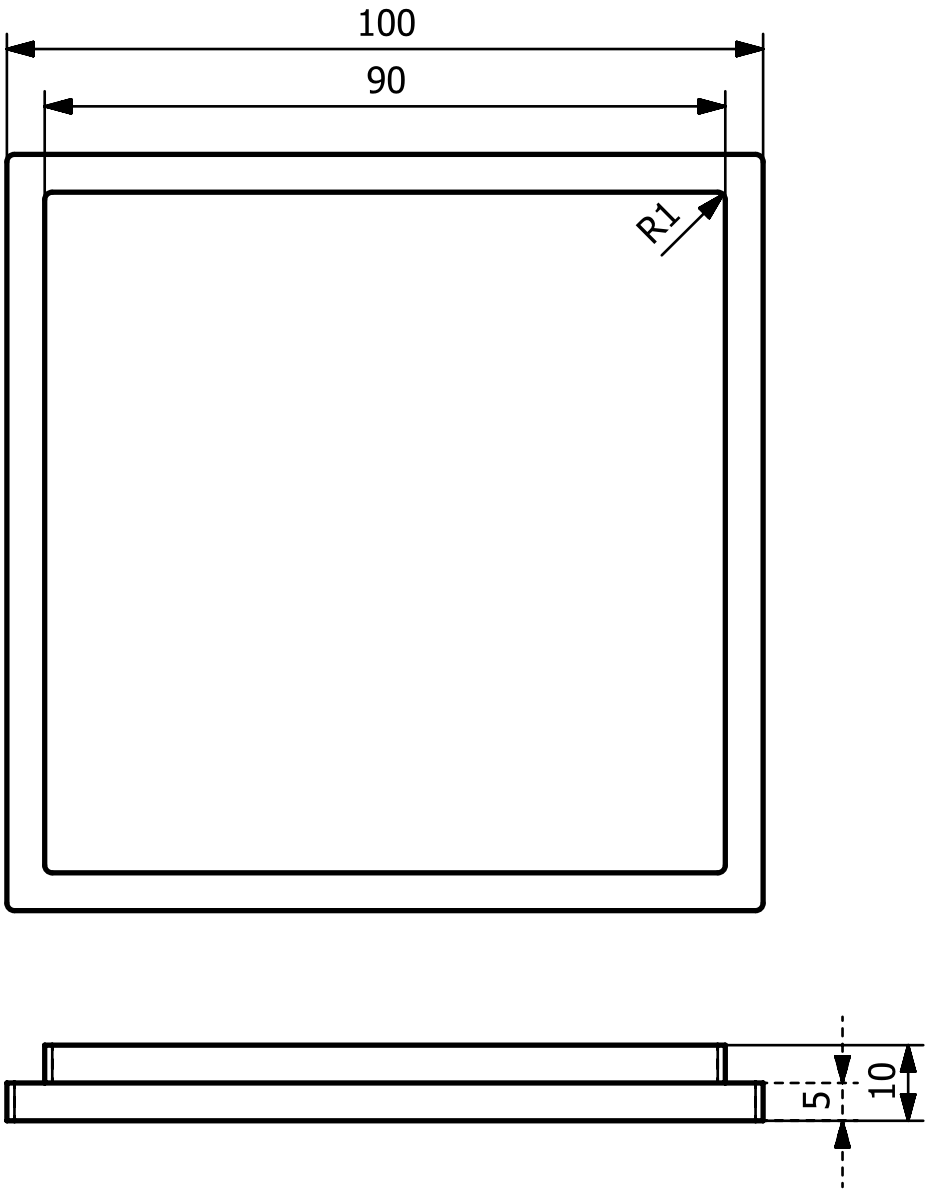
PLANO Nº: 16

AUTOR:

CONSTANTINO BELLO CORBEIRA

FIRMA:

A3



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A120

TÍTULO DEL TFG:

ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO

TÍTULO DEL PLANO:

TAPA CAJA ELECTRÓNICA

FECHA: SEP 2017

ESCALA: 1:1

PLANO Nº: 17

AUTOR:

CONSTANTINO BELLO CORBEIRA

FIRMA:

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice del documento PLIEGO DE CONDICIONES

11.5. Información sobre la construcción del sistema objeto

La base sobre la que se coloca la estructura puede ser de cualquier material, en mi caso, he utilizado una plancha de madera de contrachapado, es importante colocar los dos propulsores por fuera de la base, ya que las corrientes de aire generadas pueden generar perturbaciones en estos.

Las piezas serán ensambladas utilizando o bien los tornillos de 6-32 u un adhesivo con base de cianocrilato.

El motor que permite el movimiento en el eje Z, a pesar de este último estar bloqueado, es un motor reciclado de un radiador eléctrico, la caja de PLA que rodea dicho motor ha sido diseñada a medida para este, así como el adaptador Motor-Rótula. En caso de querer replicar este proyecto, se deberá rediseñar esta parte en función del motor que se tenga disponible.

Las conexiones eléctricas han sido llevadas a cabo utilizando o bien una crimpadora disponible en la escuela universitaria politécnica o soldadura de estaño convencional. En este último caso, están protegidas contra contactos indeseados por plástico termorretráctil.

El contrapeso se rellenará de arena convencional, se ha calculado el volumen basandose en la densidad de esta para que su peso sea idóneo.

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

ESTADO DE MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice del documento ESTADO DE MEDICIONES

11.6. Componentes estructurales

Tabla 11.6.0.1 – Lista de componentes estructurales

Componente	Referencia	Cantidad
PLA filamento 1,75mm 1kg	Rojo rubí - Pantone 485C	1
Acople de tornillo - 1/4”	ROB-12488	1
Varilla de acero lisa - 1/4”D x 3”L	ROB-12478	1
Varilla de acero lisa - 1/4”D x 9”L	ROB-0027	1
Rodamiento con pestaña - 1/4”	ROB-13012	6
Pack tornillos 6-32 - 1”	ROB-12528	9

11.7. Componentes eléctricos y electrónicos

Tabla 11.7.0.1 – Lista de componentes eléctricos y electrónicos

Componente	Referencia	Cantidad
Banana Macho Negra 4mm 20A 60V	BN-60N	1
Banana Macho Roja 4mm 20A 60V	BN-60R	1
Banana Hembra Negra 4mm 10A 60V	HB100N	1
Banana Hembra Roja 4mm 10A 60V	HB100R	1
BeagleBone Black	BBONE-BLACK-4G	1
Conector USB Hembra Aérea Tipo A	E-SU5	1
Turnigy 20AWG Silicone Wire 15m	150000148-0	1
Favourite ESC USB Programming Tool	586000003-0	1
FVT LittleBee 30A ESC	-	2
Módulo MPU-9250 9 ejes	-	1
Escorpión M-2205-2350KV (CW & CCW)	275000006-0	1
Gemfan 5030 Propellers (CW & CCW)	329000401-0	2

11.8. Mano de obra

Tabla 11.8.0.1 – Mano de obra

Tarea	Personal	Horas
Diseño en NX 11.0 de las piezas que conforman el proyecto	Graduado en Ingeniería Electrónica Industrial y Automática	30
Construcción del sistema	Graduado en Ingeniería Electrónica Industrial y Automática	20
Configuración Inicial de la Beaglebone Black	Graduado en Ingeniería Electrónica Industrial y Automática	2
Programación en Python de los algoritmos que controlan el sistema	Graduado en Ingeniería Electrónica Industrial y Automática	25
Sintonización manual de los algorítmos de control PID	Graduado en Ingeniería Electrónica Industrial y Automática	10
Investigación y búsqueda de información	Graduado en Ingeniería Electrónica Industrial y Automática	40
Redacción de la Memoria	Graduado en Ingeniería Electrónica Industrial y Automática	45

TÍTULO: **ESTABILIZACIÓN DE UN SISTEMA AEROPROPULSADO**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2017

AUTOR: EL ALUMNO

Fdo.: CONSTANTINO BELLO CORBEIRA

Índice del documento PRESUPUESTO

12 Presupuesto

12.1. Precios unitarios de materiales y mano de obra

12.1.1. Materiales

Tabla 12.1.1.1 – Coste de los materiales

Componente	Cantidad	Coste Unitario (€)	Coste (€)
PLA filamento 1,75mm 1kg	1	15.74	15.74
Acople de tornillo - 1/4”	1	5.20	5.20
Varilla de acero lisa - 1/4”D x 3”L	1	0.90	0.90
Varilla de acero lisa - 1/4”D x 9”L	1	3.75	3.75
Rodamiento con pestaña - 1/4”	6	1.60	9.60
Pack tornillos 6-32 - 1”	1	5.20	5.20
Banana Macho Negra 4mm 20A 60V	1	1.28	1.28
Banana Macho Roja 4mm 20A 60V	1	1.28	1.28
Banana Hembra Negra 4mm 10A 60V	1	0.63	0.63
Banana Hembra Roja 4mm 10A 60V	1	0.63	0.63
BeagleBone Black	1	52.14	52.14
Conector USB Hembra Aérea Tipo A	1	2.23	2.23
Turnigy 20AWG Silicone Wire 15m	1	4.96	4.96
FVT LittleBee 30A ESC	2	13.48	26.96
Favourite ESC USB Programming Tool	1	6.55	6.55
Escorpión M-2205-2350KV (CW & CCW)	1	33.19	33.19
Gemfan 5030 Propellers (CW & CCW)	2	1.58	3.16
Módulo MPU-9250 9 ejes	1	6.15	6.15
		TOTAL	179.55

12.1.2. Mano de Obra

Tabla 12.1.2.1 – Coste de la mano de obra

Tipo de mano de obra	Unidades (h)	Coste (€/h)	Coste Total (€)
Graduado en Ingeniería Electrónica Industrial y Automática	172	40	6880

12.2. Coste total

Tabla 12.2.0.1 – Coste total

Concepto	Coste (€)
Materiales	179.55
Mano de obra	6880
Total (IVA inc.)	**7059.55**